Project Testament Document

Project Name: *Tender*

Requirement 1: Callbacks

Location in Project:

MainActivity.kt, activity_main.xml

Description:

Callback functions are implemented in MainActivity.kt for user interactions with various Views defined in activity_main.xml.

1. Button Click Listeners for Like, Dislike, and Superlike:
    - In the setupFABAndButtons function, we have set click listeners for likeButton, superlikeButton, and dislikeButton. These listeners invoke the swipeCard function with different directions based on the button clicked.
    - **How to Access:** In the app, interact with the like, dislike, or superlike buttons on the main screen. Each button triggers a distinct animation and card swipe, demonstrating the callback functionality.
2. Floating Action Button for Menu:
    - The floating action button (FAB), identified as floatingActionButton, is also set with a click listener in the setupFABAndButtons function. When clicked, it shows a popup menu.
    - **How to Access:** Click on the floating action button on the main screen to open a popup menu, showing the callback action.
3. Menu Item Click Listener:
    - In the showPopupMenu function, a PopupMenu is created with menu items. The setOnMenuItemClickListener is used to handle menu item clicks, which then calls showSearchLocationFragment or showLikedDislikedFragment based on the item selected.
    - **How to Access:** After opening the popup menu using the floating action button, select an item from the menu to trigger a callback that displays a new fragment.

Requirement 2: Logging

Location in Project:

MainActivity.kt

Description:

Important events, such as API requests and responses, are logged to Logcat with unique tags and varying levels within MainActivity.kt.

How to Access:

Run the app and observe the Logcat in the development environment to see the logged events.

1. Logging in onCreate Method:
    - At the beginning and end of the onCreate method, there are log statements with the tag "MainActivity" that indicate the start and completion of the activity's creation.
    - Example: Log.d("MainActivity", "onCreate started") and Log.d("MainActivity", "onCreate finished").
2. Logging in Configuration Change Handling:
    - In the onConfigurationChanged method, logging is used to indicate the change in orientation of the device.
    - Example: Log.d("MainActivity", "Landscape mode") and Log.d("MainActivity", "Portrait mode").
3. Logging in CardStackView Setup:
    - After setting up the CardStackView in setupCardStackView, a log statement indicates the completion of this setup.
    - Example: Log.d("MainActivity", "CardStackView setup completed").
4. Logging API Responses in getYelpData:
    - When fetching data from the Yelp API in the getYelpData method, successful and failed responses are logged with different levels. Successful responses use Log.d (debug level), and errors use Log.e (error level).
    - Example: Log.d("YelpData", "Response: $responseBody") and Log.e("YelpData", "Failed to get data: ${response.message}").
5. Logging in Swipe Actions:
    - In the onCardSwiped method, there are logs for swiping actions, indicating the direction of the swipe and the action taken as a result.
    - Example: Log.d("MainActivity", "onCardSwiped: $direction").

Requirement 3: Layouts

Location in Project:

activity_main.xml

Description:

The activity_main.xml layout file prominently features the use of ConstraintLayout, satisfying the requirement to use this layout type at least once with multiple internal views. ConstraintLayout allows for creating complex and flexible UI designs with a flat view hierarchy, which improves performance. In this layout, various UI elements like TextView, Toolbar, LinearLayout, and CardStackView are positioned relative to each other and to the parent layout using constraint properties such as

- layout_constraintTop_toTopOf,
- layout_constraintBottom_toBottomOf,
- layout_constraintRight_toRightOf, and
- layout_constraintLeft_toLeftOf.

In addition to ConstraintLayout, the layout file also uses a LinearLayout container (buttonContainer), which is a requirement to use another type of layout container. The LinearLayout holds LottieAnimationView elements and is set to horizontal orientation, allowing these elements to be arranged in a row.

How to Access:

Open the main screen of the app. The use of ConstraintLayout is evident in the positioning and arrangement of various elements like the toolbar, buttons, and card stack view. The LinearLayout can be seen at the bottom part of the screen, containing the like, dislike, and superlike buttons, showcasing the horizontal arrangement of these elements.

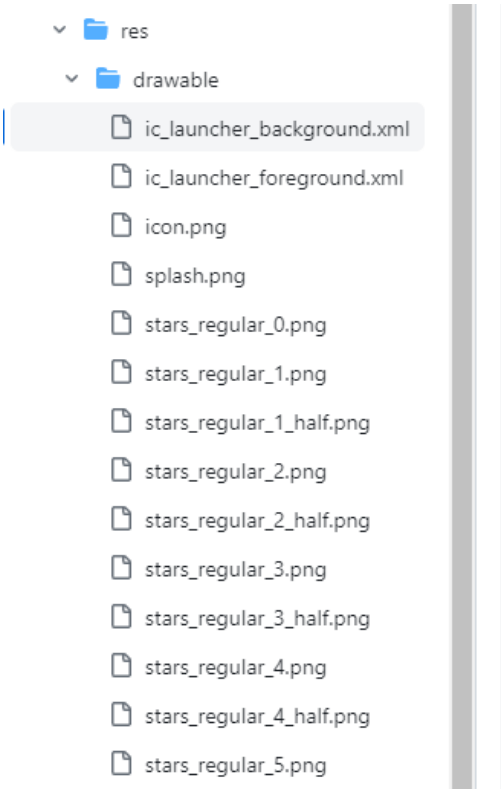
Requirement 4: Resources

Location in Project:

res/drawable, res/mipmap-*

Description:

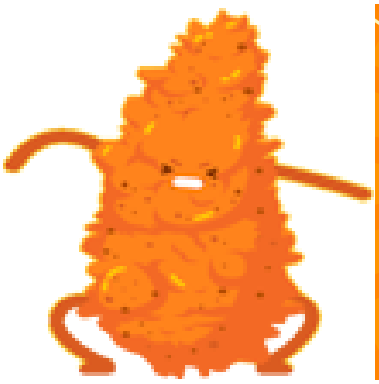
The app utilizes image resources such as icons and backgrounds, located in the drawable and mipmap directories. These images are used interactively in the UI.




```
22          android:pathData="M29,0L29,108"
23          android:strokeWidth="0.8"
24          android:strokeColor="#33FFFFFF" />
25      <path
26          android:fillColor="#00000000"
27          android:pathData="M39,0L39,108"
28          android:strokeWidth="0.8"
29          android:strokeColor="#33FFFFFF" />
30      <path
31          android:fillColor="#00000000"
32          android:pathData="M49,0L49,108"
33          android:strokeWidth="0.8"
34          android:strokeColor="#33FFFFFF" />
35      <path
36          android:fillColor="#00000000"
37          android:pathData="M59,0L59,108"
38          android:strokeWidth="0.8"
39          android:strokeColor="#33FFFFFF" />
40      <path
41          android:fillColor="#00000000"
42          android:pathData="M69,0L69,108"
43          android:strokeWidth="0.8"
44          android:strokeColor="#33FFFFFF" />
45      <path
46          android:fillColor="#00000000"
```

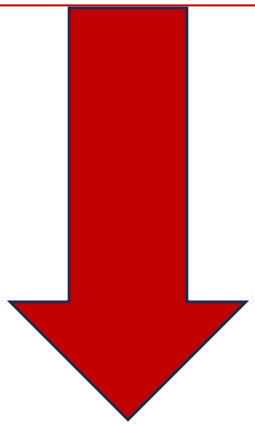
How to Access:

Observe the use of these resources in the app's UI, such as the app icon and background images.



Requirement 5: Extra Credit (Embedding Media) DID NOT ATTEMPT THIS PORTION OF EXTRA CREDIT

THERE ARE STILL MORE PAGES



[Grab your reader's attention with a great quote from the document or use this space to emphasize a key point. To place this text box anywhere on the page, just drag it.]

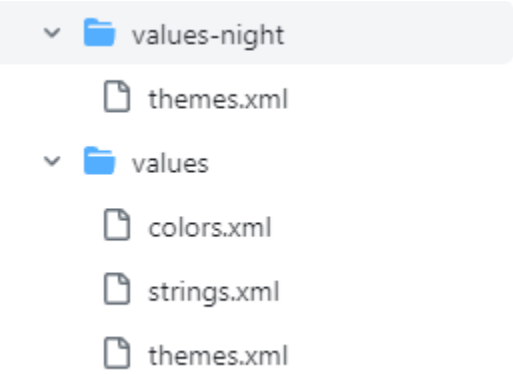
Requirement 6: Resource Qualifiers

Location in Project:

res/values/themes.xml, res/values-night/themes.xml, res/values/colors.xml, res/values/strings.xml

Description:

The project makes use of resource qualifiers to dynamically choose alternative resources based on device configuration.



1. Translations (String Resources): In values/strings.xml, string resources like app_name, no_more_restaurants, confirm, search, location, search_location, and liked_disliked are defined. To fulfill the requirement for multiple translations, similar string resources should be defined in additional values-<language_code>/strings.xml files for different languages.
2. Night Theme (Using Night Resource Qualifier): The values-night/themes.xml is used to define a night theme, which is an alternative resource based on the device's theme settings. The night theme customizes the app's appearance when the device is in night mode.
3. Color Resources: The values/colors.xml file defines various color resources used throughout the app, like colorPrimary and colorAccent. These colors can be utilized to adapt the app's appearance to different themes or styles.

How to Access:

1. To view the translations, change the device's language setting and observe the app's text change accordingly.
2. To see different layouts, rotate the device to change orientation or run the app on different screen sizes.
3. For the night theme, enable the night mode in the device's settings and observe the change in the app's theme.

4. Color changes can be observed in the app's UI where these color resources are used.

Requirement 7: Persistence

Location in Project:

MainActivity.kt

Description:

The persistence requirement in our app is addressed through several mechanisms:

1. DataStore for User Preferences:
    - The app uses DataStore (as seen with DataStoreManager.getInstance and edit function calls) to save user preferences like search terms and locations (SEARCH_TERM_KEY, LOCATION_KEY).
    - This is evident in the loadDataStoreAndRefreshData and updateSearchCriteriaDataStore functions, where preferences are retrieved, updated, and saved.
2. ViewModel for Surviving Device Rotations:
    - The handling of device orientation changes and state restoration in onConfigurationChanged and restoreInstanceState provides a mechanism to maintain UI state across configuration changes. Data like defaultSearch, defaultLocation, and isDataLoaded is preserved and restored.
3. Database for Storing Data:
    - The DBHelper class is used to interact with a local SQLite database. This is demonstrated in the refreshBusinesses function, where data is fetched from the database (dbHelper.getLikedBusinessNames()), and in the onCardSwiped method, where business details are stored (dbHelper.addOrUpdateBusiness).
4. Saving and Restoring Instance State:
    - The onSaveInstanceState and restoreInstanceState methods are used to save and restore the activity's state across configuration changes like device rotations.

How to Access:

1. DataStore Preferences: Change the search criteria in the app. Exit and relaunch the app to see the saved preferences still in effect.
2. UI State Across Rotations: Rotate the device while using the app and observe that the UI state (like the current search term and location) is maintained.

3. SQLite Database Interactions: Interact with features like liking or disliking a business. Exit and relaunch the app, then access the liked/disliked list to see the persisted data.
4. State Restoration: Perform actions in the app, rotate the device, and notice that the app's state (like the current page or selected items) is preserved and restored.

Requirement 8: User Interface

Location in Project:

java/com/example/tender/BusinessDetailsFragment.kt,

java/com/example/tender/LikedDislikedFragment.kt,

java/com/example/tender/SearchLocationFragment.kt

Description:

1. BusinessDetailsFragment: This fragment displays detailed information about a selected business. It uses a DialogFragment to present details like business name, location, reviews, and images. The fragment dynamically sets its size and incorporates a MapView for displaying the business location and a ViewPager2 with a CircleIndicator3 for image slideshows.
2. LikedDislikedFragment: This fragment shows lists of liked and disliked businesses. It uses RecyclerView for efficient display and management of these lists. The RecyclerView is set up with a LinearLayoutManager and a custom adapter (SimpleBusinessAdapter), which handles the display of business data.
3. SearchLocationFragment: This fragment allows users to input and change search criteria for businesses (search term and location). It uses EditText fields for user input and includes logic to save these preferences using DataStore.

How to Access:

1. BusinessDetailsFragment: Interact with a business card in the main screen to view its details. This action should open the BusinessDetailsFragment displaying detailed information about the business.
2. LikedDislikedFragment: Access this fragment via an option in the app, likely from the main screen or a menu. It displays the lists of liked and disliked businesses.
3. SearchLocationFragment: Access this fragment to change the search criteria. It could be opened from a button or menu option on the main screen.

Requirement 9: RESTful Interactivity

Location in Project:

MainActivity.kt

Description:

The application demonstrates RESTful interactivity by communicating with the Yelp API. This interaction is primarily conducted in the MainActivity class, where HTTP GET requests are used to fetch business data from Yelp.

1. HTTP GET Request for Business Data:
    - The method getYelpData in MainActivity executes a GET request to the Yelp API. It constructs a URL with query parameters for the search term and location, sends the request using OkHttpClient, and processes the response.
    - This method is called during the initial data load and whenever the user updates search criteria, ensuring that the app pulls fresh data from the remote API.
2. Displaying API Data:
    - The fetched data (list of businesses) is displayed in a CardStackView using a custom adapter (BusinessCardAdapter). Each card represents a business, showing its details like name, image, and rating.
3. Fetching and Displaying Detailed Business Data:
    - Detailed data for a specific business, including reviews, is fetched using the getYelpBusinessDetails method when a user selects a business card. This also involves a GET request to the Yelp API for more detailed information.

How to Access:

1. Viewing Business Cards: Launch the app to see a stack of business cards in the main screen, each representing data fetched from the Yelp API.
2. Interacting with Business Cards: Swipe through the business cards or select one to view more detailed information, indicating interaction with the Yelp API.
3. Changing Search Criteria: Use the search/location feature (likely accessed through a menu or button on the main screen) to change the search term or location, which will trigger new API requests and refresh the displayed data.