NearbyStore application

The idea of this app was a location-based store finder that allows users to search for nearby stores. It uses the Google Maps API to fetch nearby stores and display them on a map. The app also allows users to view details about each store, leave reviews, and add stores to their favorites.
For this app, Pixel 3 with API level 33 and Android 13.0 (“Tiramisu”) | x86_64 was used with the following properties:

Properties	
avd.ini.displayname	Pixel 3 API 33
avd.ini.encoding	UTF-8
AvdId	Pixel_3_API_33
disk.dataPartition.size	6442450944
fastboot.chosenSnapshotFile	
fastboot.forceChosenSnapshotBoot	no
fastboot.forceColdBoot	no
fastboot.forceFastBoot	yes
hw.accelerometer	yes
hw.arc	false
hw.audioInput	yes
hw.battery	yes
hw.camera.back	virtualscene
hw.camera.front	emulated
hw.cpu.ncore	4
hw.device.hash2	MD5:8a60718609e0741c7c0cc225f49c5590
hw.device.manufacturer	Google
hw.device.name	pixel_3
hw.dPad	no
hw.gps	yes
hw.gpu.enabled	yes
hw.gpu.mode	auto
hw.initialOrientation	portrait
hw.keyboard	yes
hw.lcd.density	440
hw.lcd.height	2160
hw.lcd.width	1080
hw.mainKeys	no
hw.ramSize	2048
hw.sdCard	yes
hw.sensors.orientation	yes
hw.sensors.proximity	yes
hw.trackBall	no
image.androidVersion.api	33
image.sysdir.1	system-images/android-33/google_apis_playstore/x86_64/
PlayStore.enabled	true
runtime.network.latency	none
runtime.network.speed	full
showDeviceFrame	yes
skin.dynamic	yes
tag.display	Google Play
tag.displaynames	Google Play
tag.id	google_apis_playstore
tag.ids	google_apis_playstore
vm.heapSize	256

This is a Java code for an Android application that uses Google Maps to display nearby stores and allows users to interact with them.

MainActivity.java

onCreate() method

The ***onCreate*** method is called when the activity is created. It initializes the UI components, sets up the Google Map, and checks for location permissions.

- It sets the title of the activity to "Maps Activity".
- It initializes the Google Map fragment and sets up the map.
- It checks if Google Maps is installed on the device and prompts the user to install it if not.
- It requests location permissions from the user.
- It sets up buttons for selecting a location, viewing favorites, and removing all favorites.

showLocationDialog Method
This method displays a dialog with buttons for selecting different locations. When a location button is clicked, it moves the map to that location and fetches nearby stores.

fetchNearbyStores() method

The fetchNearbyStores method is used to fetch nearby stores based on the selected category. It makes a GET request to the Google Maps API using Volley. The API endpoint is *https://maps.googleapis.com/maps/api/place/nearbysearch/json* and it passes the current location, radius, type (based on the selected category), and API key as parameters.

- **API Key**: The API key is required to authenticate each request to the Google Maps API. If the API key is not provided, the API returns a **REQUEST_DENIED** status with an error message indicating that an API key is required.
- **Response Handling**: When the response is received, it parses the JSON response and adds markers to the map for each store.

onMapReady() method

The onMapReady method is called when the Google Map is ready. It sets up the map settings, adds markers for favorite stores, and sets up listeners for marker clicks and map clicks.

- **Map Settings**: It sets up the map settings such as enabling zoom controls.
- **Favorite Stores**: It loads favorite stores from the database and adds markers to the map for each favorite store.
- **Marker Click Listener**: It sets up a listener for marker clicks. When a marker is clicked, it shows a dialog with more details about the store.
- **Map Click Listener**: It sets up a listener for map clicks. When the map is clicked, it adds a red marker at the clicked location and shows a dialog to enter the name of the store.

showMarkerOptions() method

This method displays a dialog with options for a marker when it is clicked. The options include viewing details, rating the store, viewing reviews, viewing ratings, leaving a review, navigating to the store, and removing the marker.

showStoreDetails() method

The showStoreDetails method is used to show a dialog with more details about the store when a marker is clicked. It shows a dialog with the store name, address, rating, and opening hours. It also provides options to view reviews and leave a review.

leaveReview() method

The leaveReview() method is used to leave a review for a store. It shows a dialog to enter the review, and when the review is submitted, it stores the review in the database.

viewReviews() method

The viewReviews() method is used to view reviews for a store. It loads reviews from the database and shows a dialog with all the reviews for the store.

navigateToStore() method

The navigateToStore method is used to navigate to a store using the Google Maps app. It creates a Uri for the Google Maps app with the current location and the store location, and then starts an intent to open the Google Maps app.

removeMarker() method

The removeMarker method is used to remove a marker from the map and the database. It removes the marker from the map and then removes the store from the database.

Error Handling

The app handles errors such as when the Google Maps API returns a REQUEST_DENIED status due to the absence of an API key. It also handles exceptions that might occur when interacting with the database.

API Key

The API key is required to authenticate each request to the Google Maps API. If the API key is not provided, the API returns a *REQUEST_DENIED* status with an error message indicating that an API key is required.

In order to obtain an API Key, detailed instructions can be found in the following link:
https://developers.google.com/maps/documentation/embed/get-api-key

In order to restrict the API key to specific Android applications, a debug certificate fingerprint or a release fingerprint should be provided with the following commands:
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android

After the generation of the fingerprint, the package name and the SHA-1 certificate fingerprint can be added in the Android restrictions of the https://console.cloud.google.com/apis/credentials/key

The API restrictions which specify the enabled APIs that this key can call are the following:

- Maps SDK for Android
- Maps JavaScript API
- Places API (New)
- Places API
- Geolocation API
- Google Cloud APIs
- Time Zone API
- Directions API
- Geocoding API
- Route Optimization API
- Roads API

The enabling of the services for the Google Maps project can also be conducted after opening a terminal in the gcloud and running the following command:

```
gcloud services enable \
  --project "PROJECT_ID" \
  "directions-backend.googleapis.com" \
  "distance-matrix-backend.googleapis.com" \
  "geocoding-backend.googleapis.com" \
  "geolocation.googleapis.com" \
  "maps-backend.googleapis.com" \
  "maps-embed-backend.googleapis.com" \
  "places-backend.googleapis.com" \
  "routes.googleapis.com"
```

This command enables the necessary services for Google Maps Platform, including:

- Directions API
- Distance Matrix API
- Geocoding API
- Geolocation API
- Maps JavaScript API
- Maps Embed API
- Places API
- Routes API

These services are required for device location detection, navigation, and geolocation features.

General Operation of the Application

- The application starts with a map displaying the user's current location (issue: "*Unable to locate"* message is thrown)
- The user can select a different location from a dialog.
- The application fetches nearby stores using the Google Places API and adds markers to the map.
- The user can click on a marker to view options for that store.
- The user can view details, rate, leave a review, view reviews, view ratings, navigate to, or remove a store.
- The application stores the user's favorite stores in a database.
- The user can view and manage their favorite stores.

DatabaseHelper.java

This DatabaseHelper class is a SQLite database helper for managing a database in an Android application. It extends the SQLiteOpenHelper class, which is a helper class for managing database creation and version management.

Constructor

The constructor DatabaseHelper (Context context)initializes the database helper with a context and a database name. It checks if the database exists, and if not, it creates the database.

Database Existence Check

TheDatabaseExists(Context context) method checks if the database file exists in the context's database path.

Key Features and Methods:

Database Creation and Upgrades:

- The onCreate method is called when the database is created. It creates three tables: favorites, reviews, and ratings.
- The onUpgrade method is called when the database version is changed. It drops the existing tables and recreates them.

Favorites Management:

- addStoreToFavorites(Store store): Adds a store to the favorites table.
- getFavoriteStores(): Retrieves all stores from the favorites table.
- removeStoreFromFavorites(String storeName): Removes a store from the favorites table and its associated reviews and ratings.

Reviews Management:

- *addReview(String storeName, String review):* Adds a review for a store to the reviews table.
- *getReviewsForStore(String storeName):* Retrieves all reviews for a store from the reviews table.

Ratings Management:

- *addRating(String storeName, double rating):* Adds a rating for a store to the ratings table.
- *getRatingForStore(String storeName):* Retrieves the rating for a store from the ratings table.

Database Cleanup:

- *removeAllFavorites()*: Removes all data from the favorites, reviews, and ratings tables.

Key Class Variables:

- *DATABASE_NAME*: The name of the database.
- *DATABASE_VERSION*: The version of the database.

Key Methods Explanation:

- ***databaseExists(Context context)***: Checks if the database file exists.
- ***onCreate(SQLiteDatabase db):*** Called when the database is created. It creates the tables.
- ***onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion):*** Called when the database version is changed. It drops the existing tables and recreates them.
- ***addRating(String storeName, double rating):*** Adds a rating for a store to the ratings table.
- ***getRatingForStore(String storeName):*** Retrieves the rating for a store from the ratings table.
- ***removeAllFavorites():*** Removes all data from the favorites, reviews, and ratings tables.
- ***addReview(String storeName, String review):*** Adds a review for a store to the reviews table.
- ***getReviewsForStore(String storeName):*** Retrieves all reviews for a store from the reviews table.
- ***addStoreToFavorites(Store store):*** Adds a store to the favorites table.
- ***getFavoriteStores():*** Retrieves all stores from the favorites table.

- ***removeStoreFromFavorites(String storeName):*** Removes a store from the favorites table and its associated reviews and ratings.

onCreate() Method

The **onCreate(SQLiteDatabase db)** method is called when the database is created. It creates two tables: **favorites** and **reviews**. The favorites table has columns for **id, name**, **latitude**, and **longitude**, while the **reviews** table has columns for **id**, **store_name** and **review**.

onUpgrade() Method

The **onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)** method is called when the database version is upgraded. It drops the existing tables and recreates them.

addReview() Method

The **addReview(String storeName, String review)** method adds a review to the reviews table. It inserts a new row with the store name and review.

removeStoreFromFavorites() method

The *removeStoreFromFavorites(String storeName)* method removes a store from the `favorites` table. It deletes the row with the matching store name.

getReviewsForStore Method

The *getReviewsForStore (String storeName)* method retrieves all reviews for a specific store from the `reviews` table. It returns a list of MyReview objects.

addStoreToFavorites() method

The *addStoreToFavorites (Store store)* method adds a store to the favorites table. It inserts a new row with the store's name, latitude, and longitude.

getFavoriteStores() method

The *getFavoriteStores()* method retrieves all favorite stores from the favorites table. It returns a list of Store objects.

Error Handling

The class handles exceptions and errors during database operations by printing the stack trace.

FavoritetoresActivity.java

This class is an Android activity that displays a list of favorite stores. Here's a detailed explanation of the class:

Class Overview

The FavoriteStoresActivity class is an Android activity that extends AppCompactActivity. It is responsible for displaying a list of favorite stores retrieved from a database.

Class Variables

- databaseHelper: An instance of DatabaseHelper used to interact with the database.
- FavoriteListVIew: A ListView that displays the list of favorite stores.

onCreateMethod()

The **onCreate()** method is called when the activity is created. It initializes the activity's UI components and loads the favorite stores from the database.

initializeViews() method

The **initializeVIews()** method initializes the UI components of the activity. It finds the ListView and Button components by their IDs and sets up a click listener for the button.

loadFavoriteStores() method

The **loadFavoriteStores()** method loads the favorite stores from the database using the DatabaseHelper instance. It retrieves the list of favorite stores, creates a StoreAdapter to bind the data to the ListView, and sets up an item click listener for the ListView.

Item Click Listener

When an item in the ListView is clicked, the activity navigates back to the MainActivity with the selected store's location as an extra in the intent.

Error Handling

The class handles exceptions that might occur when loading the favorite stores from the database. If an exception occurs, it displays a toast message with the error message.

StoreListActivity.java

This class is an Android activity that displays a list of stores. Here's a detailed explanation of the class:

Class Overview

The StoreListActivity class is an Android activity that extends AppCompactActivity. It is responsible for displaying a list of stores.

Fields

- **storeListView**: A ListView that displays the list of stores.

onCreate Method

The onCreate method is called when the activity is created. It initializes the activity's UI components and sets up the list view.

List View Setup

The method retrieves the list of stores from the intent extras using *getParcelableArrayListExtra(“stores”)* It then creates a StoreAdapter instance and sets it as the adapter for the list view using storeListView.setAdapater(adapter).

Intent Extras

The activity expects a list of Store objects to be passed as an intent extra with the key "stores". This list is used to populate the list view.

Error Handling

The method uses an assert statement to ensure that the list of stores is not null. If the list is null, the app will crash with an AssertionError.

Log Statement

There is no log statement in this class.

Parcelable

The Store class must implement Parcelable to be passed as an intent extra.

Activity Flow

The activity is started with an intent that includes a list of Store objects. The activity displays the list of stores in a list view.

MyReview.java class

This class represents a review for a store. Here's a detailed explanation of the class:

Class Overview

The MyReview class is a simple Java class that represents a review for a store. It has a single field `text` to store the text of the review.

Constructor

The constructor *MyReview (String rating, String text)* initializes a new MyReview object with the provided rating and text. However, the rating parameter is not used in the constructor, which might be an oversight.

Field

- text: A String field to store the text of the review.

Getter Method

- *getText()*: getter method to retrieve the text of the review.

Purpose

The purpose of this class is to encapsulate the details of a review, specifically the text of the review. It can be used to store and retrieve reviews for stores in the app.

Store.java class

This class represents a store with a name and a location specified by latitude and longitude. It implements the Parcelable interface, which allows instances of the class to be written to and read from a Parcel, a container for a message (data and object references) that can be sent through an IBinder.

Class Overview

The Store class is a simple Java class that represents a store with a name and a location. It has three fields: name, latitude, and longitude which are initialized through the constructor.

Fields

- *name*: A String field to store the name of the store.
- *latitude*: A double field to store the latitude of the store's location.
- *longitude*: A double field to store the longitude of the store's location.

Constructor

The constructor ***Store (String name, double latitude, double longitude)*** initializes a new *Store* object with the provided name, latitude, and longitude.

Getter Methods

- *getName():* A getter method to retrieve the name of the store.
- *getLatitude():* getter method to retrieve the latitude of the store's location.
- *getLongitude():* A getter method to retrieve the longitude of the store's location.

Parcelable Implementation

The class implements the Parcelable interface, which allows instances of the class to be written to and read from a Parcel. This is useful for passing objects between activities or services in Android.

- *describeContents()*: A method that returns an integer describing the contents of the parcel.
- *writeToParcel(Parcel dest, int flags)* A method that writes the object's data to the parcel.
- *CREATOR*: A static field that holds a Parcelable. Creator instance, which is used to create new instances of the class from a parcel.

StoreAdapter.java class

This class is a custom adapter for a ListView that displays a list of stores. Here's a detailed explanation of the class:

Class Overview

The StoreAdapter class extends BaseAdapter and is used to bind a list of Store objects to a ListView. It provides the necessary methods to populate the ListView with the store data.

Fields

- **context**: A Context object that is used to inflate the list item layout.
- **stores**: A **List** of **Store** objects that is used to populate the **ListView**.

Constructor

The constructor **StoreAdapter (Context context, List<Store> stores)** initializes a new StoreAdapter object with the provided **context** and **stores**.

Adapter Methods

The class overrides the necessary methods of BaseAdapter to provide the data for the ListView:

- **getCount():** Returns the number of items in the list.
- **getItem(int position):** Returns the item at the specified position.
- **getItemId(int position):** Returns the ID of the item at the specified position.
- **getView(int position, View convertView, ViewGroup parent):** Returns the view for the item at the specified position.

getView() method

The *getView* method is responsible for creating and populating the view for each item in the list. It:

1. Inflates the list item layout if the convertView is null.
2. Retrieves the Store object at the specified position.
3. Sets the text of the nameTextView to the store's name.
4. Returns the populated view.

INTERFACES

The provided code consists of several XML files that define the user interface and resources for an Android app. Here's a detailed explanation of each file:

activity_favorite_stores.xml

This file defines the layout for the FavoriteStoresActivity. It includes:

- Three buttons to view cafe, health, and restaurant favorites.
- A ListView to display the list of favorite stores.
- A "Back to Map" button.

activity_main.xml

This file defines the layout for the MainActivity.
This layout is a user interface designed for an Android application. It is defined using XML and utilizes the ConstraintLayout from the AndroidX library. Here's a breakdown of the components and their properties:

ConstraintLayout:
- This is the root layout element.
- It has a width and height set to match_parent, meaning it will occupy the full screen.

Fragment:
- This is a SupportMapFragment from the Google Maps API.
- It is used to display a map.
- It also has a width and height set to match_parent, so it will fill the entire screen.

Button:
- This is a button with the text "Select Location".
- It has a width set to wrap_content, meaning it will only be as wide as its content.
- It has a height set to wrap_content, meaning it will only be as tall as its content.
- It has a margin of 16dp at the top and bottom.

Spinner:
- There are two spinners: *store_category_spinner* and *filter_spinner*.

- Both have a width and height set to wrap_content.
- They are positioned using constraints:
- store_category_spinner is constrained to the top of filter_spinner.
- filter_spinner is constrained to the top of view_favorites_button.

LinearLayout:

- This is a vertical linear layout.
- It has a width set to wrap_content and a height set to wrap_content.
- It has a blue-gray background color (#ADD8E6).
- It is constrained to the bottom of the parent and aligned to the left and right of the parent.

Buttons Inside LinearLayout:

There are two buttons inside the linear layout:

- view_favorites_button with the text "View Favorites".
- remove_all_favorites_button with the text "Remove All Favorites".

Both buttons have a width set to wrap_content and a height set to wrap_content.

In summary, this layout includes a map, a button to select a location, two spinners for selecting categories and filters, and a linear layout with two buttons at the bottom. The layout is designed to be responsive and adapt to different screen sizes

activity_maps.xml

This file defines the layout for the MapsActivity. It includes:

Root Element:

- The root element is a LinearLayout with the following attributes:
- xmlns:android: This is the namespace for Android-specific XML attributes.
- xmlns:app: This is the namespace for attributes specific to the app.
- xmlns:tools: This is the namespace for tools-specific XML attributes.
- android:layout_width and android:layout_height: These specify the width and height of the layout to be the full width and height of the parent, respectively.
- android:fitsSystemWindows: This attribute ensures that the layout fits within the system windows.
- android:orientation: This specifies the orientation of the layout as vertical.

Toolbar:

- The first child element is an androidx.appcompat.widget.Toolbar with the following attributes:
- android:id: This specifies the ID of the toolbar.
- android:minHeight: This sets the minimum height of the toolbar to the standard action bar size.
- android:layout_width and android:layout_height: These specify the width and height of the toolbar.
- app:titleTextColor: This sets the text color of the toolbar title to white.
- android:background: This sets the background color of the toolbar to the color defined as colorPrimary.

Spinner and Other Views:

- The second child element is a LinearLayout with a horizontal orientation. It contains:
- A Spinner with the ID filter_spinner. This is used to filter items in the list.
- There is a comment suggesting that other views, such as a search bar or buttons, can be added here.

ListView:

- The third child element is another LinearLayout with a vertical orientation. It contains:
- A ListView with the ID list_places. This is used to display a list of places.

The layout is structured to have a toolbar at the top, followed by a horizontal layout containing a spinner and potentially other views, and finally a vertical layout containing a list view. This layout is likely used to display a list of places with filtering capabilities.

The XML namespaces (xmlns:android, xmlns:app, and xmlns:tools) are used to define the prefixes for attributes specific to Android, the app, and tools, respectively. These namespaces are used to avoid conflicts between attributes with the same name from different sources.

activity_store_list.xml

This file defines the layout for the StoreListActivity. It includes:

XML Declaration:

- <?xml version="1.0" encoding="utf-8"?>: This is the standard XML declaration, specifying the version and encoding of the document.

LinearLayout:

- *<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android":*
- *xmlns:android:* This attribute specifies the namespace for Android-specific XML attributes. The URL *http://schemas.android.com/apk/res/android* is used to define the prefix android for attributes.
- *android:orientation="vertical":* This sets the orientation of the LinearLayout to vertical, meaning the child elements will be stacked vertically.
- *android:layout_width="match_parent"* and *android:layout_height="match_parent":* These set the width and height of the LinearLayout to match the width and height of its parent, respectively.

ListView:

- *<ListView android:id="@+id/store_list_view":*
- *android:id="@+id/store_list_view"*: This assigns an ID to the ListView so it can be referenced in the Java code.
- *android:layout_width="match_parent"*: This sets the width of the ListView to match the width of its parent.
- *android:layout_height="0dp*": This sets the height of the ListView to 0dp, which is used in conjunction with android:layout_weight to distribute the available space.
- *android:layout_weight="1":* This specifies that the ListView should take up all the available space in the vertical direction.

TextView:

- *<TextView android:id="@+id/store_details_text_view":*
- *android:id="@+id/store_details_text_view"*: This assigns an ID to the TextView so it can be referenced in the Java code.
- *android:layout_width="match_parent":* This sets the width of the TextView to match the width of its parent.
- *android:layout_height="wrap_content"*: This sets the height of the TextView to wrap its content, meaning it will be as tall as the text it contains.
- *android:textSize="16sp":* This sets the text size to 16 scale-independent pixels.
- *android:textStyle="bold"*: This sets the text style to bold.

In summary, this layout consists of a vertical LinearLayout containing a ListView and a TextView. The ListView takes up all the available space, and the TextView is placed below it with a fixed height based on its content.

list_item_store.xml

This file defines the layout for each item in the ListVIew of stores. It includes:

XML Declaration:
<?xml version="1.0" encoding="utf-8"?>: This is the standard XML declaration, specifying the version and encoding of the document.

LinearLayout:

- <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android":
  xmlns:android: This attribute specifies the namespace for Android-specific XML attributes.
- The URL http://schemas.android.com/apk/res/android is used to define the prefix android for attributes.
- xmlns:app: This attribute specifies the namespace for app-specific XML attributes.
- xmlns:tools: This attribute specifies the namespace for tools-specific XML attributes. The URL http://schemas.android.com/tools is used to define the prefix tools for attributes.
- android:layout_width="match_parent" and android:layout_height="match_parent":

These set the width and height of the LinearLayout to match the width and height of its parent, respectively.

- android:fitsSystemWindows="true": This attribute ensures that the layout fits within the system windows.
- android:orientation="vertical": This sets the orientation of the LinearLayout to vertical, meaning the child elements will be stacked vertically.

TextViews:

There are three TextView elements:

- <TextView android:id="@+id/name_text_view">:
- android:id="@+id/name_text_view": This assigns an ID to the TextView so it can be referenced in the Java code.
- android:layout_width="match_parent": This sets the width of the TextView to match the width of its parent.
- android:layout_height="wrap_content": This sets the height of the TextView to wrap its content, meaning it will be as tall as the text it contains.
- android:textSize="16sp": This sets the text size to 16 scale-independent pixels.
- android:textStyle="bold": This sets the text style to bold.
- <TextView android:id="@+id/address_text_view"> and <TextView android:id="@+id/reviews_text_view">:These have similar attributes, but with different IDs and text sizes.

In summary, this layout consists of a vertical LinearLayout containing three TextView elements. Each TextView has a unique ID and different text sizes.

colors.xml

This file defines the color resources used in the app. It includes:

XML Declaration:

<?xml version="1.0" encoding="utf-8"?>: This is the standard XML declaration, specifying the version and encoding of the document.

Resources:

<resources>: This is the root element, which contains all the color resources.

Color Resources:

There are seven color resources defined:

- <color name="colorPrimary">#3F51B5</color>:
- name: This specifies the name of the color resource, which can be used in the application's layout files.
- #3F51B5: This is the hexadecimal color code for the colors

The other color resources are defined similarly:

- colorPrimaryDark: #303F9F
- colorAccent: #FF4081
- primary_text: #333333
- green: #00FF00
- white: #FFFFFF
- primary_color: #2196F3

These color resources can be used throughout the application to maintain consistency in the visual design. For example, you can use **@color/colorPrimary** in a layout file to set the color of a view to the primary color defined here. The error messages in the provided sources (,,) indicate that the URLs for the Android schemas are not resolvable, which is expected since these URLs are not meant to be accessed directly. They are used as namespaces in XML files to define the prefixes for Android-specific attributes.

ids.xml

This file defines the IDs for the views used in the app. It includes:

XML Declaration:

<?xml version="1.0" encoding="utf-8"?>: This is the standard XML declaration, specifying the version and encoding of the document.

Resources:
<resources>: This is the root element, which contains all the resource items.

Resource Items:

There are seven resource items defined:
<item name="map" type="id" />:

- name: This specifies the name of the resource item, which can be used in the application's layout files.

- type: This specifies the type of the resource item, which in this case is an ID.

The other resource items are defined similarly:

- view_favorites_button
- remove_all_favorites_button
- favorite_button
- store_list_view
- list_view
- back_button

strings.xml

This file defines the string resources used in the app. It includes:

XML Declaration:

<?xml version="1.0" encoding="utf-8"?>: This is the standard XML declaration, specifying the version and encoding of the document.

Resources:

<resources>: This is the root element, which contains all the string resources.

String Resources:

There are six string resources defined:

- <string name="app_name">Nearest Stores</string>:
- name: This specifies the name of the string resource, which can be used in the application's code.
- Nearest Stores: This is the value of the string resource.

The other string resources are defined similarly:

- title_activity_maps: Nearest Stores
- action_settings: Settings
- action_geolocate: Pick Place
- back_to_map: Back to Map
- search_hint: Search for nearby stores

These string resources can be used throughout the application to maintain consistency in the text displayed to the user. For example, you can use @string/app_name in a layout file to set the text of a view to the application name defined here.

styles.xml

This file defines the styles used in the app. It includes:

Resources:

<resources>: This is the root element, which contains all the resource items.

Style:

<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">:
name: This specifies the name of the style, which is AppTheme.

parent: This specifies the parent theme, which is Theme.AppCompat.Light.DarkActionBar. This means that AppTheme inherits all the attributes from Theme.AppCompat.Light.DarkActionBar.

Style Items:

There are three style items defined:

- <item name="colorPrimary">@color/colorPrimary</item>:
- name: This specifies the name of the style item, which is colorPrimary.
- @color/colorPrimary: This references a color resource named colorPrimary.

The other style items are defined similarly:

- colorPrimaryDark: @color/colorPrimaryDark
- colorAccent: @color/colorAccent

Unresolved Issues in the current application:

There seems to be an issue with the ability to locate the current device, throwing an error that the app is unable to locate and the navigation does not work as a result.

Potential feature expansion:

The potential feature expansions of this app include:

1. **Integration with Artificial Intelligence (AI)**: The app could leverage AI to improve the user experience, such as providing more personalized and relevant search results, enhancing local discovery, and generating more accurate and detailed information about locations.
2. **Enhanced Navigation Experience**: The app could redesign its navigation UI to make it more intuitive and modern, such as using rounded corners, buttons to share locations, and a smaller expanded card that doesn't take up the whole screen.
3. **Increased Use of Internet of Things (IoT) Data**: The app could utilize IoT data from connected sensors to provide more real-time and accurate information about locations, making the maps more "live" and dynamic.
4. **More Immersive and Realistic Visuals**: The app could incorporate more immersive and realistic visuals, such as 3D elements, to enhance the user experience and provide a more detailed view of locations.
5. **More Personalized Recommendations**: The app could provide more personalized recommendations based on user preferences and behavior, such as suggesting hidden gems and undiscovered businesses.
6. **Improved Search Capabilities**: The app could improve its search capabilities by using natural language processing and machine learning to understand user queries and provide more accurate and relevant results.
7. **Enhanced Business Profiles and Listings**: The app could provide businesses with more detailed and accurate profiles, including information about their offerings, ambiance, and unique aspects, and allow them to engage more effectively with users.
8. **Increased Focus on Sustainability and Accessibility**: The app could incorporate features that promote sustainability and accessibility, such as providing information about electric vehicle charging locations, public transportation options, and accessible routes.