COURSE PROJECT: An Android app to introduce Chinese culture

Student ID: 20170933

Student Name: Mukaddam Khamraeva

1. Introduction

The aim of this project is to design and implement an Android app inspired by the popular Pinterest app, which focuses on sharing and discovering pictures related to Chinese culture. The app provides users with a platform to explore various aspects of Chinese culture through captivating images, organized into categories such as cities, cuisines, and festivals. By leveraging the power of visual representation, the app aims to foster a deeper understanding and appreciation of Chinese culture among users.

2. Project Inspiration

The app takes inspiration from the well-known Pinterest app, which has revolutionized the way users discover and share visual content. By adopting a similar concept, the Chinese culture introduction app aims to create an engaging and visually appealing experience for users. By leveraging the app's intuitive interface and image-centric approach, users can seamlessly navigate through different categories, search for specific images, and explore related photos to delve deeper into their areas of interest.

3. App Design

The app's design focuses on simplicity, user-friendliness, and aesthetically pleasing visuals. The landing page serves as an enticing entry point, featuring an elegant app title and a captivating background image that sets the tone for the Chinese cultural exploration journey. The app supports both portrait and landscape layouts, ensuring a seamless user experience regardless of device orientation. By incorporating intuitive navigation and visually appealing thumbnails, the list page enables users to browse through a curated collection of images within each category. The detail page provides users with a deeper insight into specific images, allowing them to explore related photos and gain a richer understanding of the cultural context.

3.1 Landing Page (SplashActivity)

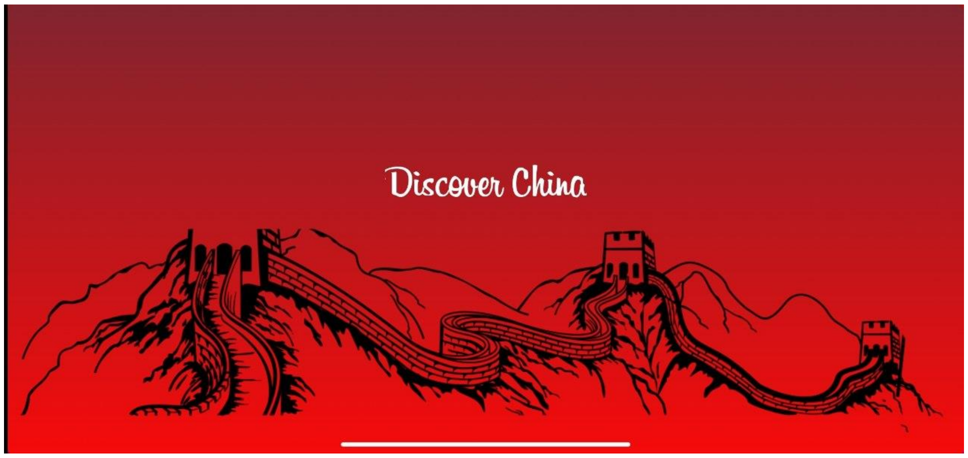
The layout for the landing page is designed to create a visually appealing and immersive experience for the users.

The layout consists of a **ConstraintLayout** as the root view, which allows for flexible positioning and sizing of the contained elements. The background of the landing page looks simple and elegant without any extra pictures or animation, it is defined using a custom gradient, represented by the **app_main_gradient** drawable. This gradient combines shades of red and blue, creating a visually striking effect.

The central focus of the landing page is the **LinearLayout,** this layout is vertically oriented and uses the **center_horizontal** and **bottom** gravity attributes to center its child views both horizontally and at the bottom of the layout. This arrangement ensures that the app title and image are prominently displayed.

The **TextView** within the inner **LinearLayout** is responsible for displaying the app name. It uses a custom font, **billabong**, which adds a unique and stylish touch to the text. The font size is set to 40sp, ensuring that the app name is visually appealing and easy to read.

Below the app name, the landing page includes an **ImageView** that showcases a sketch of the Chinese Great Wall. The image adds visual interest and serves as a symbolic representation of China's rich cultural heritage and historical significance.



3.2 List Page (HomeFragment)

3.2.1 fragment_home.xml:

The HomeFragment layout represents a screen that displays a list of items related to different aspects of Chinese culture. Let's go through the structure and functionality of the HomeFragment:

- The main content of the page is contained within a **NestedScrollView**, allowing for smooth scrolling of the content. This view occupies the available height and is wrapped inside a **LinearLayout**, which allows for vertical stacking of the child views.

- Within the **NestedScrollView**, there's a nested **LinearLayout** with a vertical orientation. This layout acts as a container for the individual sections of the HomeFragment.
- At the top of the page, there's a horizontal **LinearLayout** that contains four **TextView** elements representing different categories: Cities, Cuisine, Festivals, and Kung Fu. These categories provide options for users to explore various aspects of Chinese culture.
- Each **TextView** has a custom background defined by the **border_round_all_dark** drawable, giving them a consistent visual style. They also have click listeners associated with them, allowing users to select a category of interest.
- Following the category section, there's a **RecyclerView** component with the id **recyclerPins**. This RecyclerView is responsible for displaying the list of items related to the selected category. The **HomeAdapter** class is used as the adapter for this RecyclerView.

3.2.2 HomeAdapter:

The **HomeAdapter** is a custom adapter class that extends **RecyclerView.Adapter** and is used for populating the items in the **RecyclerView** on the HomeFragment. Here's an overview of the key aspects of this adapter:

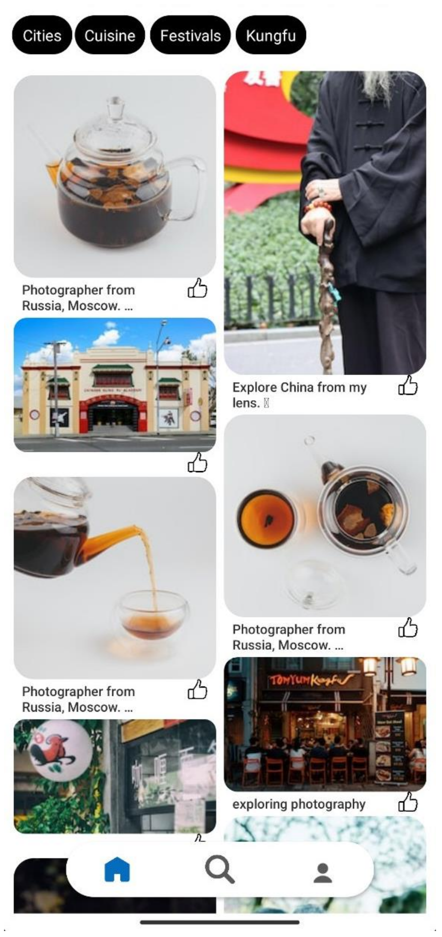
- The adapter takes a **HomeFragment** instance and an **ArrayList** of **Photo** items as input in its constructor.
- The **onCreateViewHolder** method inflates the item layout defined in **item_home_view.xml** and returns a new instance of the **ViewHolder** class.
- The **onBindViewHolder** method binds the data from the **Photo** item at the given position to the corresponding views in the **ViewHolder**. It uses Glide library to load the thumbnail image, and sets the description text accordingly.
- The **ViewHolder** class holds references to the individual views within an item layout. It assigns these views using **findViewById** in its constructor.
- The **HomeAdapter** also handles click events for the like button within each item. It toggles the like status and updates the UI accordingly using **PrefsManager** to store the liked status.
- Lastly, the **callDetails** method is responsible for starting the **DetailsActivity** when an item is clicked. It passes the list of photos and the clicked position as extras in the intent.

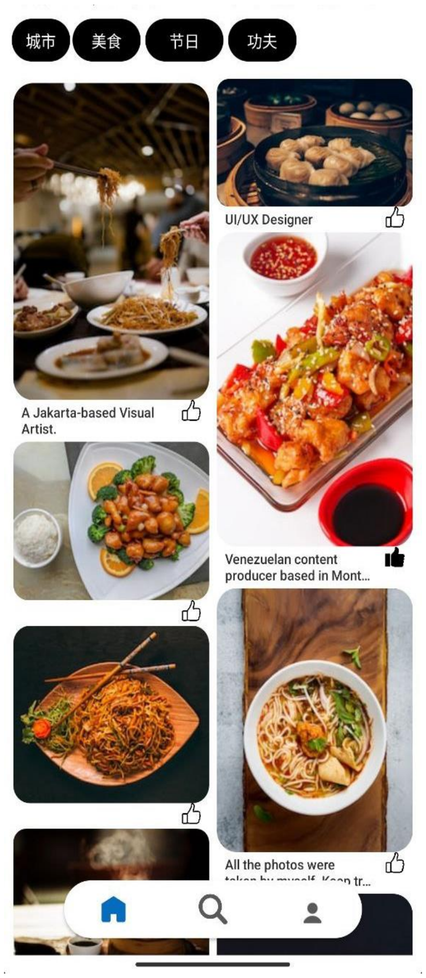
3.2.3 HomeFragment:

The **HomeFragment** handles the initialization of views, API calls, and the interaction with the user. Here's an overview of its key features:

- In the **onCreateView** method, the fragment inflates the layout defined in **fragment_home.xml** and initializes the views using **initViews** method.

- The **initViews** method sets up the **RecyclerView** with a **StaggeredGridLayoutManager** and applies a **SpacesItemDecoration** to add spacing between the items.
- The category **TextViews** (Cities, Cuisine, Festivals, Kung Fu) are assigned click listeners to handle user selection. Based on the selected category, an API call is made using **RetrofitHttp** to fetch the relevant photos.
- The **apiGetChinese** method performs the API call and updates the **RecyclerView** adapter with the fetched items by calling **refreshAdapter**.
- The **refreshAdapter** method creates a new instance of **HomeAdapter** and sets it as the adapter for the **RecyclerView**.


Cities
Cuisine
Festivals
Kungfu
Photographer from Russia, Moscow. ...
Explore China from my lens.
Photographer from Russia, Moscow. ...
Photographer from Russia, Moscow. ...
exploring photography


城市
美食
节日
功夫
UI/UX Designer
A Jakarta-based Visual Artist.
Venezuelan content producer based in Mont...
All the photos were

3.3 Details Page (DeatailsFragment)

3.3.1 **DetailsFragment** class, which is responsible for displaying the detail page of a photo. Here's the key components and their functionalities:

The fragment layout is defined in the XML file **fragment_details.xml**, it consists of several views, including **ImageView**, **TextView**, **RatingBar**, and **RecyclerView**, arranged in a nested structure. Also, back button to go back to the list page.

In the **onCreate()** method, the fragment initializes the **DetailsAdapter** and retrieves related photos using the **apiRelatedPhotos()** method.

The **DetailsAdapter** is responsible for populating the related photos in the **RecyclerView**. It is updated with the related photo list using the **addPhotos()** method.

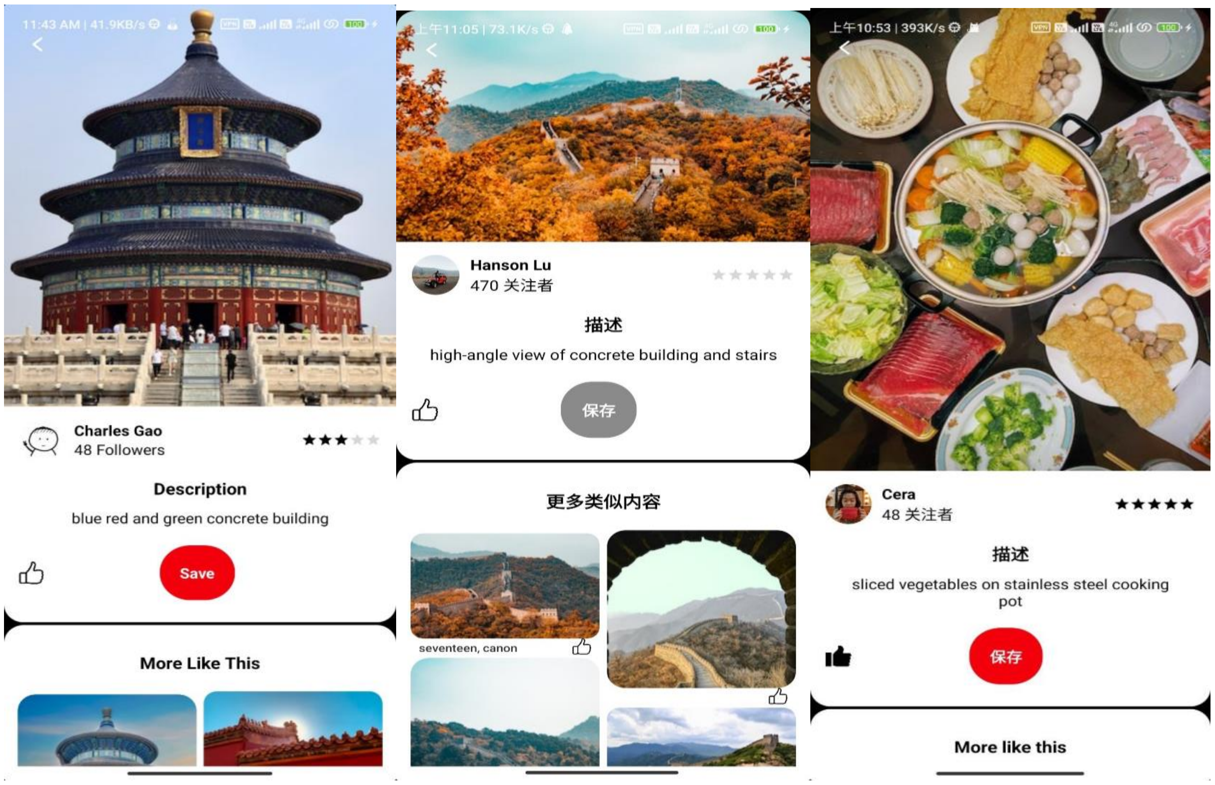
The **apiRelatedPhotos()** method makes an API call using Retrofit to fetch related photos based on the photo ID. The response is handled in the **onResponse()** method, where the retrieved photo list is added to the adapter using **addPhotos()**. If the photo list is empty, " Related photos not found "message is displayed in the **tvRelated** text view.

The **getDescription()** method is a helper method that determines the appropriate description for the photo. It checks the **alt_description**, **description**, and user's name to return a suitable description string.

In the **DetailsFragment**, the **PrefsManager** is used to save and retrieve the rating value for a specific photo. When the user changes the rating using the **RatingBar**, the new rating value is saved in the shared preferences using the photo's ID as the key. The saved rating value is then retrieved and applied to the **RatingBar** when the **DetailsFragment** is recreated.

Similarly, the **PrefsManager** is used to save and retrieve the like status of photos. When the user clicks on the like button (**iv_btn_like**), the like status is toggled and saved in the shared preferences using the photo's ID as the key. The saved like status is then retrieved and used to update the UI (e.g., changing the like button's icon) when the **DetailsFragment** is recreated.

Overall, the **DetailsFragment** class manages the UI and functionality of the detail page, including displaying the photo, user information, description, related photos, and handling actions like saving and liking the photo.



3.4 Language

I created resource files for each supported language English and Chinese, and I provided translations for the strings in each **strings.xml** file. App switches to different version according to system language.

3.5 Background music (**BackgroundMusicService)**

I created separate class for background music and applied in it in parent activity so the music keeps playing in all activities.

1. Service Setup:
    - The **BackgroundMusicService** class extends the **Service** class, which allows it to run in the background.
    - The **MediaPlayer** instance **mediaPlayer** is created in the **onCreate()** method, using the audio resource **R.raw.background_music** to initialize it. The background music is set to loop continuously.
2. Starting and Stopping the Music:
    - The **onStartCommand()** method is called when the service is started. It starts playing the background music using **mediaPlayer.start()**. The **isMusicPlaying** flag is set to **true** to indicate that the music is currently playing.

   - The **onDestroy()** method is called when the service is destroyed. It stops the music playback using **mediaPlayer.stop()**, releases the **MediaPlayer** resources using **mediaPlayer.release()**, and sets the **isMusicPlaying** flag to **false**.
3. Service Control:
   - The **onBind()** method is not implemented, as the service does not provide binding capabilities.
   - The **IBinder** returned by **onBind()** is **null**.
   - The **isMusicPlaying()** method returns the value of the **isMusicPlaying** flag, indicating whether the music is currently playing.
   - The **stopMusic()** method sets the **isMusicPlaying** flag to **false**, allowing external components to stop the music if needed.
   - The **startMusic()** method is a static helper method that can be called to start the background music service. It takes a **Context** parameter and starts the service using an **Intent**. The **isMusicPlaying** flag is set to **true** after starting the service.

5. Implementation

The app is developed using Java and Android SDK, following the best practices learned throughout the course. The core code emphasizes modularization and maintainability, employing design pattern Model-View-Controller (MVC) to ensure a clean and scalable architecture. Key implementation aspects include efficient image loading and caching, responsive UI design for different screen sizes, seamless integration of search functionality, and retrieval of related images based on user preferences. Additionally, the app incorporates language support for both English and Chinese, seamlessly adapting to the system language settings.

5.1 BaseActivity

The **BaseActivity** class serves as the parent activity for all other activities within the app. It provides a set of common functionalities and behaviors that can be shared across multiple activities, promoting code reusability and maintainability.

The **BaseActivity** class includes the following key features:

5.1.1 Background Music Management

One of the functionalities provided by the **BaseActivity** is the management of background music. By extending the **BaseActivity**, child activities can easily start and stop the background music through the **startBackgroundMusic()** and **stopBackgroundMusic()** methods. This promotes a consistent and seamless user experience, as the background music continues to play across different activities.

5.1.2 Multilingual Support

Another important feature of the **BaseActivity** is its support for multiple languages. It handles the language switching based on the system language, allowing the app to adapt to the user's preferred language. This is achieved through the **attachBaseContext()** method, which sets the language context for the activity.

5.1.3 Lifecycle Management

The **BaseActivity** class also manages the lifecycle events of the activities. By overriding the **onStart()** and **onStop()** methods, it ensures that the background music is started when the activity comes into the foreground and stopped when the activity goes into the background or is no longer visible to the user. This prevents the unnecessary playback of background music and optimizes system resources.

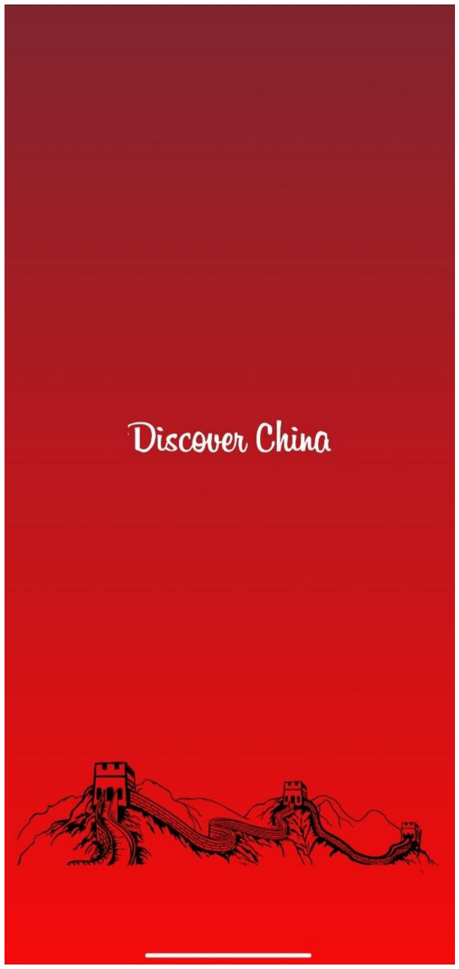
By centralizing these common functionalities in the **BaseActivity**, developers can avoid repetitive code implementation and easily maintain consistency across different activities. It promotes clean code architecture and enhances the overall development efficiency of the app.

5.2 SplashActivity

SplashActivity extends the **BaseActivity** class, which provides the common functionalities used throughout the app. The **SplashActivity** sets the window flags to enable full-screen display, sets a transparent status bar, and applies a gradient background.

The **initViews()** method initializes the necessary views, and the **countDownTimer()** method starts a countdown timer. After the timer finishes, the **callHomeActivity()** method is called to navigate the user to the home activity (**HomeActivity**) of the app.

The **setTransparentStatusBar()** method is responsible for setting the status bar color as transparent and applying a gradient background to the entire window.



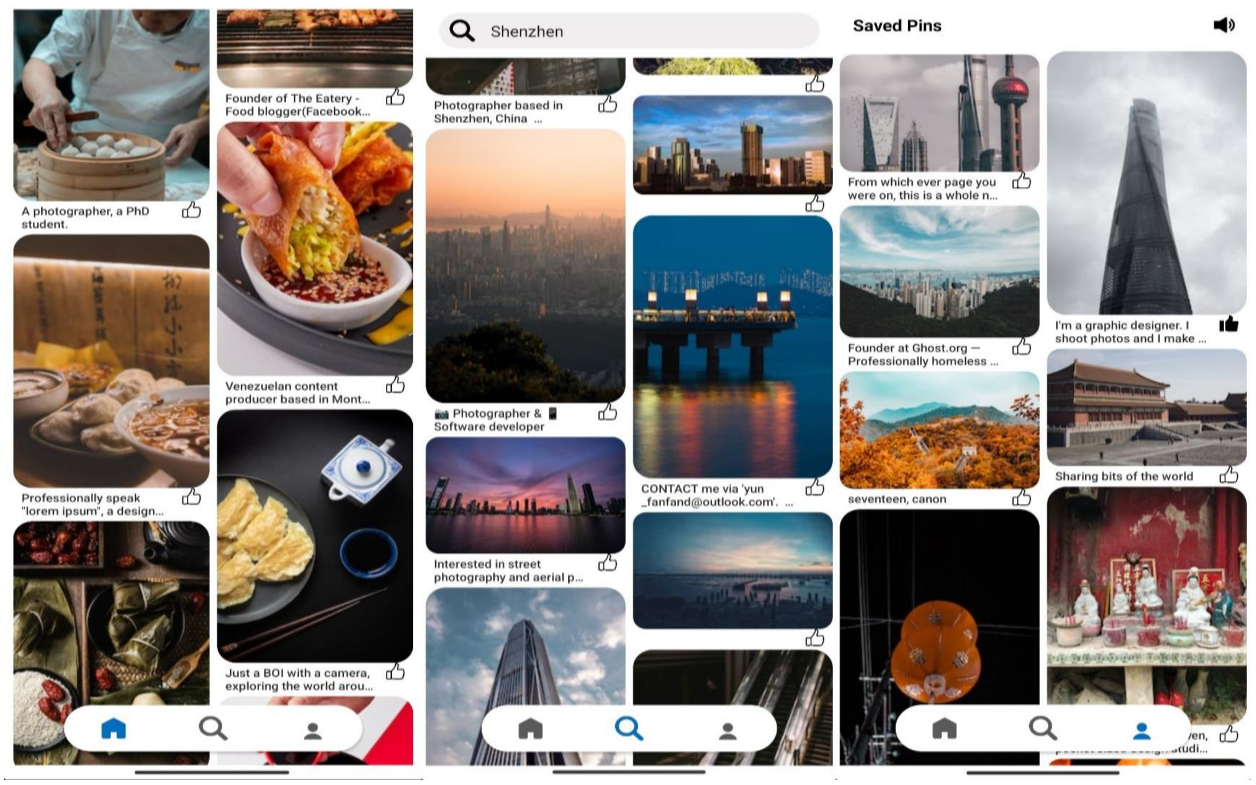
5.3 HomeActivity

The **HomeActivity** class serves as the main activity for my app. Here's an explanation of the code:

4. Activity Setup:
   - The **HomeActivity** extends **BaseActivity**, indicating that it inherits functionality from the **BaseActivity** class.
   - In the **onCreate()** method, the layout file **activity_main.xml** is set as the content view for the activity using **setContentView(R.layout.activity_main)**.
   - Instances of the **HomeFragment**, **SearchFragment**, and **ProfileFragment** are created.
5. Fragment Navigation:
   - The **bottomNavigationView** and **flFragment** are initialized by finding their corresponding views in the layout using **findViewById()**.
   - The **replaceFragment()** method is called with the **homeFragment** to initially show the **HomeFragment** when the activity is created.

    - A listener is set on the **bottomNavigationView** to handle navigation item selection events. Depending on the selected item, the **replaceFragment()** method is called with the corresponding fragment (**homeFragment**, **searchFragment**, or **profileFragment**) to replace the current fragment in the **flFragment** container.
6. Fragment Management:
    - The **replaceFragment()** method takes a **Fragment** as a parameter and is responsible for replacing the current fragment in the **flFragment** container.
    - It uses the **FragmentManager** to manage the fragment transactions. It first tries to pop the back stack to check if the desired fragment already exists in the back stack. If not, it creates a new fragment transaction, replaces the current fragment with the desired fragment using **ft.replace()**, adds the transaction to the back stack using **ft.addToBackStack()**, and commits the transaction using **ft.commit()**.
7. Back Button Handling:
    - The **onBackPressed()** method is overridden to handle the back button press. If the back stack has only one entry (meaning only the initial fragment is present), the activity is finished. Otherwise, the super method is called to handle the back button press as usual.

In this **HomeActivity** class, you can navigate between fragments using the bottom navigation view and replace the fragments in the **flFragment** container.


A photographer, a PhD student.
Founder of The Eatery - Food blogger(Facebook...
Venezuelan content producer based in Mont...
Professionally speak "lorem ipsum", a design...
Just a BOI with a camera, exploring the world arou...
Shenzhen
Photographer based in Shenzhen, China ...
📷 Photographer & 📱 Software developer
Interested in street photography and aerial p...
CONTACT me via 'yun _fanfand@outlook.com', ...
Saved Pins
From which ever page you were on, this is a whole n...
Founder at Ghost.org — Professionally homeless ...
seventeen, canon
I'm a graphic designer. I shoot photos and I make ...
Sharing bits of the world

5.4 DeatailsActivity

In **DetailsActivity** the **DetailsPagerAdapter** class is used as an adapter to manage the fragments displayed within a **ViewPager**. It extends the **FragmentPagerAdapter** class and provides functionality for adding fragments to the adapter and retrieving them based on their positions. Here's an explanation of its implementation:

- It extends the **FragmentPagerAdapter** class, which is a specialized pager adapter for managing fragments within a **ViewPager**.
- The class maintains a list of fragments using an **ArrayList<Fragment>** called **fragments**.
- The constructor initializes the adapter by calling the superclass constructor with the provided **FragmentManager**.
- The **addFragment()** method allows adding fragments to the adapter. It simply adds the given fragment to the **fragments** list.
- The **getCount()** method overrides the superclass method and returns the size of the **fragments** list, indicating the number of fragments managed by the adapter.
- The **getItem()** method overrides the superclass method and retrieves the fragment at the specified position from the **fragments** list.

In the **DetailsActivity**, the **DetailsPagerAdapter** is used to populate the **ViewPager** with fragments representing individual photos. The **refreshAdapter()** method in **DetailsActivity** iterates through the list of photos and adds a corresponding **DetailsFragment** for each photo to the **DetailsPagerAdapter** using the **addFragment()** method. This allows each photo to be displayed as a separate fragment within the **ViewPager**.

By using the **DetailsPagerAdapter**, the **DetailsActivity** achieves seamless navigation between different fragments representing photos within the **ViewPager** UI component.

5.5 Database

I have implemented a database using the Room library to handle the persistence of photos in ProfileFragment. The key component responsible for managing the database operations is the **PinRepository** class:

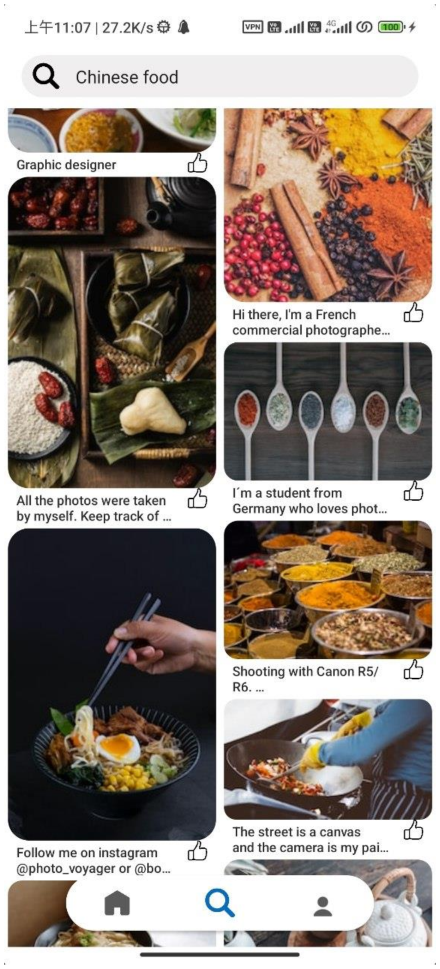
1. **Database Management:**
    - The **RoomManager** class is utilized to manage the Room database instance. It is initialized within the **PinRepository** constructor using the **RoomManager.getInstance(application)** method.
    - The **RoomManager** class abstracts the underlying complexities of the Room library and provides a streamlined interface for accessing the database.
2. **Data Access Object (DAO):**
    - The **PinDao** interface serves as the Data Access Object for the **Pin** entity, enabling database operations related to photo management.
    - By calling **dp.pinDao()**, we obtain an instance of the **PinDao** interface from the **RoomManager**.
    - The **PinDao** interface defines methods for saving, retrieving, and deleting **Pin** objects.
3. **Database Operations:**
    - The **PinRepository** class encapsulates the database operations related to photo management.
    - The **savePhoto(Pin pin)** method allows us to save a **Pin** object representing a photo to the database. It utilizes the **pinDao.savePhoto(pin)** method to accomplish this.
    - To retrieve all saved photos, we use the **getAllSavedPhotos()** method. It invokes **pinDao.getAllSavedPhotos()** and returns a list of **Pin** objects representing the saved photos.
4. **Integration with Fragments:**
    - The database implementation is integrated into the application's fragments, specifically the **DetailsFragment** and **ProfileFragment**.
    - The **DetailsFragment** includes a save button that triggers the **savePhoto(Pin pin)** method of the **PinRepository**, allowing users to save photos to the database.
    - The **ProfileFragment** retrieves the saved photos by calling **getAllSavedPhotos()** from the **PinRepository** and displays them accordingly.

5.5 Network

I have implemented network communication using the Retrofit library:

8. **RetrofitHttp Class:**
    - The **RetrofitHttp** class is responsible for setting up the Retrofit instance and providing access to the network services.
    - It contains constants for server URLs for development and production environments (**SERVER_DEVELOPMENT** and **SERVER_PRODUCTION**).
    - The **server()** method determines the appropriate server URL based on a boolean flag **IS_TESTER**, which indicates whether the application is in a testing environment.
    - The Retrofit instance is built using **retrofit2.Retrofit.Builder()**, specifying the base URL obtained from **server()**, and adding a Gson converter factory.
    - The **PhotoService** is obtained from the Retrofit instance using **retrofit.create(PhotoService.class)**, allowing access to the network API defined in the **PhotoService** interface.
9. **PhotoService Interface:**
    - The **PhotoService** interface defines the endpoints and HTTP methods for interacting with the photo-related API.
    - It includes the **ACCESS_KEY** constant, representing the authorization key required for accessing the API.
    - The interface provides the following methods:
        - **getRelatedPhotos(String id)**: Retrieves related photos for a given photo ID. It uses the **@GET** annotation with the path parameter **photos/{id}/related**.
        - **searchPhoto(String query, int page, int per_page)**: Performs a search for photos based on a query string. It uses the **@GET** annotation with the path parameter **search/photos**.
    - The methods in the interface are annotated with **@Headers** to include the authorization header with the access key.
10. **Integration with Application:**
    - The network classes, specifically the **RetrofitHttp** class and the **PhotoService** interface, are integrated into the application's functionality.
    - The **PhotoService** instance obtained from **RetrofitHttp** is used in various parts of the application (HomeFragment, DetailsFragment, SearchFragment) to make API calls for retrieving related photos and performing photo searches.
    - The API responses can be handled using Retrofit's **Call** interface, which allows for asynchronous execution and provides methods for handling success and error scenarios.

I used **Unsplash** server to get all the photos. The **RetrofitHttp** class and **PhotoService** interface facilitate the interaction with the photo-related API endpoints, allowing us to retrieve related photos and perform photo searches based on user input.


上午11:07 | 27.2K/s
Chinese food
Graphic designer
Hi there, I'm a French commercial photographe...
All the photos were taken by myself. Keep track of ...
I´m a student from Germany who loves phot...
Shooting with Canon R5/ R6. ...
The street is a canvas and the camera is my pai...
Follow me on instagram @photo_voyager or @bo...

6. Conclusion

In conclusion, the Android app developed for introducing Chinese culture combines the visual appeal of the Pinterest app with a focus on showcasing captivating images and providing users with an immersive cultural experience. By enabling users to explore and share Chinese culture through pictures, the app aims to foster a deeper appreciation and understanding of the rich heritage and traditions. Throughout the development process, various challenges were encountered and overcome, resulting in a well-designed and functional app that successfully fulfills its purpose of promoting Chinese culture and all the requirements of the project.