技术方案文档

1-项目概述

1.1 项目背景

- 抖音图文业务，通过抖音双列/及可颂产品以内容与兴趣做好社区的人与人连接；
- 产品形态上，以双列图文视频混排，主要是双列屏效比更高
- 图文体裁具有信息密度高、可控感强的优势

1.2 预期目标

制作一个简易版可颂App，实现双列作品列表页与图文详情页

从用户视角思考，在实现基础功能要求上，进一步完善App的功能

2-需求分析

2.1 功能需求

2.1.1 基础功能

- 应用框架
  - 底部导航栏
    - [x] ~~tab选项为：首页、朋友、相机(+)、消息、我（个人主页）~~
    - [x] ~~“首页”和“我”支持点击，其余无需支持点击~~
    - [x] ~~“我”页面内容以“简洁”为基调自由发挥~~
    - [x] ~~启动App后，默认在“首页”~~
  - 首页Tab
    - [x] ~~顶部Tab为“北京”“团购”“关注”“社区”“推荐”“搜索Icon”~~
    - [x] ~~默认在“社区”~~
- 双列瀑布流
  - 双列框架
    - [x] ~~下拉刷新~~
    - [x] ~~上滑LoadMore~~

- [x] ~~首刷请求失败空态页面~~
- [x] ~~刷新&加载过程的Loading提示~~

- 作品卡片
  - [x] ~~作品封面，裁切规则：宽高比限制在3:4~4:3之间~~
  - [x] ~~作品标题，优先展示标题，没有展示正文，最长显示为2行，超出使用...进行截断~~
  - [x] ~~底部：作者头像&昵称，点赞Icon&数量，~~
    - [x] ~~点赞icon&数量需要支持点击，点击后切换为已点赞状态，点赞状态采用本地磁盘持久化存储~~

- 详情页
  - 顶部作者区（常驻）
    - [x] ~~包含返回Icon，作者头像&昵称，关注按钮~~
    - [x] ~~关注按钮支持点击切换关注状态，采用本地磁盘持久化存储~~
  - 底部交互区（常驻）
    - [x] ~~快捷评论框，点赞，评论，收藏，分享~~
    - [x] ~~点赞，分享支持交互，其余不可支持~~
  - 横滑容器
    - [x] ~~使用首图作为容器比例，宽高比限制在3:4~16:9之间~~
    - [x] ~~非首图展示按首图的比例，进行裁切，确保图片内容充满容器~~
    - [x] ~~支持手动横滑切换图片~~
    - [x] ~~支持判断级别的加载态，失败态~~
  - 进度条
    - [x] ~~形状为条状~~
    - [x] ~~展示规则：单图不展示、多图展示~~
    - [x] ~~支持跟随图片切换横向滚动~~
    - [x] ~~无需支持拖动快速预览~~
  - 标题区
    - [x] ~~完整展示不截断~~
  - 正文区
    - [x] ~~正文完整展示不截断~~
    - [ ] 话题词：高亮展示，支持点击（点击页面自由发挥，简洁为主）
  - 发布日期

- [x] ~~24h内，显示具体时间，例如HH:mm，昨天HH:mm~~
- [x] ~~7天内，显示为x天前~~
- [x] ~~其余显示为具体日期，例如MM-dd~~

2.1.2 进阶功能

- 页面转场-自定义转场动画
  - 进场
    - [ ] 点击卡片封面图片逐渐放大至详情页横滑容器区域
    - [x] ~~详情页横滑容器外区域渐显~~
  - 退场
    - [x] ~~横滑容器图片逐渐缩小至双列卡片封面区域~~
    - [x] ~~详情页横滑容器外区域渐隐~~
  - 侧滑跟手退场
- 详情页
  - 背景音乐
    - [x] ~~静音按钮，添加静音icon，点击静音按钮能够静音~~
    - [x] ~~打开详情页时静音态能够记忆，初始非静音~~
    - [x] ~~关闭页面，打开其他页面，退后台时暂停播放，返回详情页回前台时恢复播放~~
    - [x] ~~支持app生命周期内生效，继承上次切换后的状态~~
    - [x] ~~app冷启后重置~~

性能需求

安全需求

3-整体方案/架构设计

整体架构模式：MVVM + 模块化设计

项目采用了MVVM（Model-View-ViewModel）架构模式，结合模块化设计，实现了清晰的分层架构：

代码块

```
app/src/main/java/com/example/myapplication/
├── model/ (数据模型层)
│   ├── Post.java - 帖子数据模型
│   └── PostDetail.java - 帖子详情数据模型
├── viewmodel/ (业务逻辑层)
```

```
6   │   ├── PostViewModel.java - 帖子列表业务逻辑
7   │   ├── PostDetailViewModel.java - 帖子详情业务逻辑
8   │   └── MineViewModel.java - 个人中心业务逻辑
9   ├── view/ (UI展示层)
10  │   ├── activity/
11  │   │   ├── MainActivity.java - 主容器Activity
12  │   │   └── PostDetailActivity.java - 帖子详情页
13  │   └── fragment/
14  │       ├── HomeFragment.java - 首页Fragment
15  │       ├── MineFragment.java - 我的页面Fragment
16  │       └── PostDetailFragment.java - 帖子详情Fragment（备用）
17  ├── repository/ (数据仓库层)
18  │   └── PostRepository.java - 数据获取和管理
19  ├── adapter/ (适配器层)
20  │   ├── PostAdapter.java - 帖子列表适配器
21  │   └── ImagePagerAdapter.java - 图片轮播适配器
22  └── util/ (工具类)
23      └── MusicPlayerManager.java - 音乐播放管理器
```

核心架构组件详解

数据流架构

代码块

```
1  网络/本地数据 → Repository → ViewModel → LiveData → View
```

各层职责划分

Model层：

- `Post`：定义帖子数据结构，包含互动数据字段
- 采用Java Bean模式，提供完整的getter/setter方法

Repository层：

- `PostRepository`：统一的数据访问层
- 负责JSON数据解析、本地缓存管理
- 实现数据获取、解析、存储的完整流程

ViewModel层：

- 继承AndroidViewModel，管理UI状态和业务逻辑
- 使用LiveData实现数据与UI的响应式绑定

- 处理用户交互逻辑和状态管理

View层：

- Activity作为容器，Fragment作为内容展示单元
- 使用观察者模式监听ViewModel的数据变化
- 处理用户界面交互和导航

关键架构特性

- 响应式数据
- 模块化设计
- 导航架构

数据管理架构

数据获取流程

- 网络请求：通过Repository获取远程数据
- JSON解析：解析服务器返回的JSON数据
- 本地缓存：使用SharedPreferences存储用户状态
- 状态同步：ViewModel管理本地状态与服务器数据的同步

缓存策略

- 使用文件缓存存储帖子数据
- SharedPreferences存储用户偏好设置（关注、点赞状态）
- 支持离线数据展示

UI架构设计

UI架构设计

列表展示

- 使用RecyclerView + StaggeredGridLayoutManager实现瀑布流布局
- `PostAdapter` 处理复杂的item布局和数据绑定
- 支持下拉刷新和上拉加载更多

交互设计

- 点击事件通过接口回调处理

- 状态变化实时更新UI
- 平滑的转场动画和状态切换

架构优势分析

可维护性

- 清晰的职责分离：各层职责明确，便于维护
- 模块化设计：功能模块独立，便于扩展
- 标准化接口：统一的数据访问和UI更新接口

可测试性

- ViewModel可独立测试业务逻辑
- Repository可测试数据获取和解析
- 依赖注入便于Mock测试

可扩展性

- 易于添加新的功能模块
- 支持多种数据源接入
- 灵活的UI组件组合

4-功能模块设计

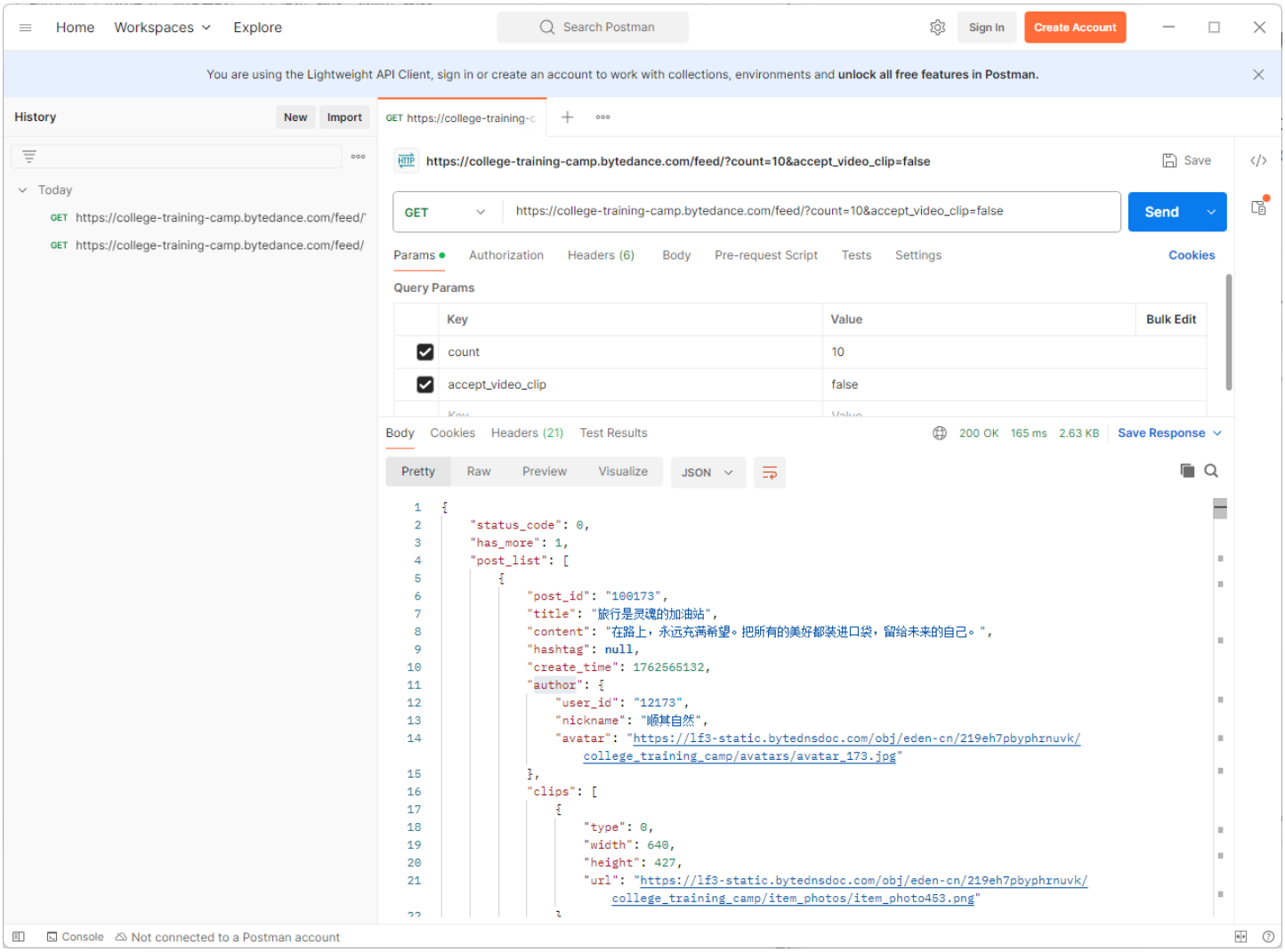
按照figma搭建页面框架

实现服务接口的连接，并正确显示首页

首先尝试使用GET请求携带query参数访问接口，如下所示访问成功并返回了

- status_code
- has_more
- post_list，一个post包含的字段就有
  - post_id
  - title
  - content
  - hashtag

- create_time
- 作者的字段author
  - user_id
  - nickname
  - avatar
- clips
  - type
  - width
  - height
  - url
- music
  - volume
  - seek_time
  - url



为了适配该接口，先对Post模型进行修改，使其符合上述字段，并添加内部类Hashtag, Author, Clip, Music

然后在HomeFragment中实现网络请求来获取真实数据

主要是OncreateView，创建FetchDataFromApi,parseJsonResponse，PostViewHolder这几个函数

需要用到Gson依赖用于处理JSON，在dependencies中添加，

可以看到使用旧版本请求已经叉掉了，提示：

a. 确保您的Android项目启用了网络权限
b. 对于Android 9.0（Pie）及以上版本，您可能需要添加网络安全配置以允许HTTP请求
c. 在实际项目中，**建议使用更现代的网络请求库如Retrofit或OkHttp的协程支持**
d. 图片加载使用了Glide库，需要确保已添加相应依赖

因为要显示真实数据的图片，修改item_post_card.xml，

在AndroidManifest.xml中添加网络权限

报错解决：

在PostDetailFragment.java的456行中，因为我们修改了Post模型中作者名获取方法为getAuthorName，因此需要替换post.getAuthor，

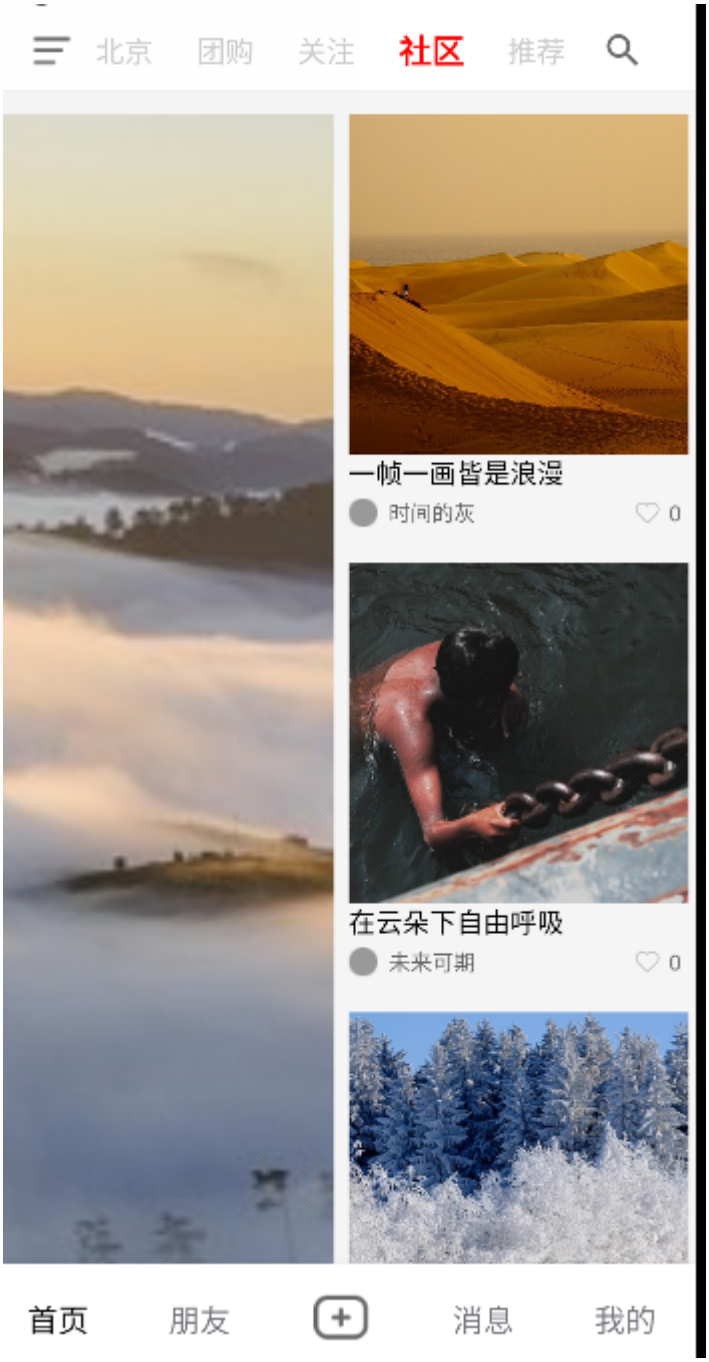
解决问题：app第一次打开进入，发现能够获取10条数据，但是只通过模拟初始化了10条数据，

然后在获取数据fetchdataFromApi以及解析json数据函数中进行了调试，发现了问题，报错：

```
代码块
1   java.lang.ClassCastException: com.google.gson.JsonNull cannot be cast to
    com.google.gson.JsonArray
```

这说明是在解析json数据时遇到了问题试图将一个null值转换为JsonArray，从而导致了有时候只能显示一条或者都是模拟数据。修改解析json数据函数，完善对null值得校验，解决该问题。

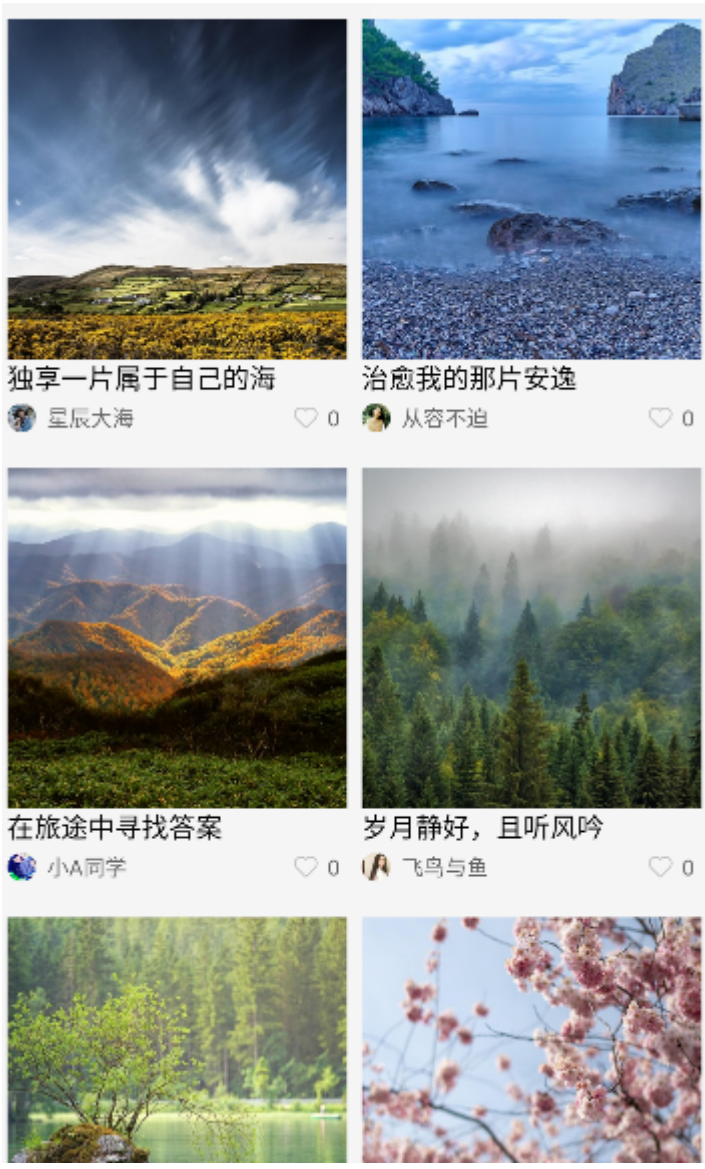
解决问题：author中包含的avatar并没有在界面中正确显示，



通过debug发现了原因是因为我的item_post_card.xml中的用户头像的设置设置了tint背景导致了显示为灰色头像，删除该背景即可：

```
代码块
    <!-- 作者头像 -->
    <!--    <ImageView-->
    <!--        android:id="@+id/author_avatar"-->
    <!--        android:layout_width="16dp"-->
    <!--        android:layout_height="16dp"-->
    <!--        android:shape="oval"-->
    <!--        android:src="@drawable/user_avatar"-->
    <!--        android:tint="#999999" />-->
    <!--    移除灰色tint-->
    <ImageView
        android:id="@+id/author_avatar"
        android:layout_width="16dp"
        android:layout_height="16dp"
        android:shape="oval"
        android:src="@drawable/user_avatar" />
```

显示效果如下：



首页点击卡片跳转到帖子详情

解决问题：遇到点击卡片详情，只能显示正确的标题，其他信息还没有正确显示。

在PostDetailFragment.java中设置逻辑，如果post不为空，则不选用模拟数据，而是调用getPostDetailFromServer函数。

解决问题：卡片详情中，用户头像和用户上传的图片不能正确显示，

首先用户头像还是可以拿到头像url了，就可以和HomeFragement中设置用户头像的方法一样，调用Glide设置圆形头像。添加loadAuthorAvatar方法，如果post不为空则获取用户头像，反之则设置默认模拟头像。

然后是图片展示，这里要改成调用数据中的clips数据展示，原来的代码为：

代码块

```
private class ImagePagerAdapter extends RecyclerView.Adapter<ImageViewHolder> {
    private List<Integer> images;

    public ImagePagerAdapter(List<Integer> images) {
        this.images = images;
    }

    @NonNull
    @Override
    public ImageViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_image_page, parent, false);
        return new ImageViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ImageViewHolder holder, int position) {
        holder.bind(images.get(position));
    }

    @Override
    public int getItemCount() {
        return images.size();
    }
}
```

修改后的代码为：修改PostDetailFragment.java，先获取第一张图片，计算宽高比，然后设置容器的宽高比，以及对应的布局文件

```java
// 修改setupData方法，添加头像加载逻辑
private void setupData() {
    if (postDetail == null) return;

    // 设置作者信息
    authorName.setText(postDetail.getAuthorName());

    // 使用Glide加载真实的作者头像
    loadAuthorAvatar();

    // 检查是否已关注该作者
    checkFollowStatus();

    // 设置图片适配器
    setupImageAdapter();

    // 设置帖子内容
    postTitle.setText(postDetail.getTitle());
    postContent.setText(postDetail.getContent());

    // 设置话题标签
    setupTopics();

    // 设置发布日期
    postDate.setText(formatDate(postDetail.getPublishDate()));

    // 设置点赞状态
    updateLikeStatus(isLiked);
    // 显示点赞数
    likeCount.setText(String.valueOf(currentLikeCount));
}

// 添加加载作者头像的方法
private void loadAuthorAvatar() {
    if (authorAvatar == null || getArguments() == null) return;

    // 获取原始Post对象以获取头像URL
    Post post = (Post) getArguments().getSerializable(ARG_POST);
    if (post != null && post.getAuthor() != null &&
post.getAuthor().getAvatar() != null) {
        String avatarUrl = post.getAuthor().getAvatar();
        try {
            // 使用Glide加载头像，确保圆形显示
            Glide.with(this)
                .load(avatarUrl)
                .placeholder(R.drawable.user_avatar) // 加载占位图
                .error(R.drawable.user_avatar) // 错误占位图
```

```
                .circleCrop() // 圆形裁剪
                .into(authorAvatar);
        } catch (Exception e) {
            // 加载失败时使用默认头像
            setupDefaultAvatar();
        }
    } else {
        // 无头像信息时使用默认头像
        setupDefaultAvatar();
    }
}

// 提取默认头像设置为单独方法
private void setupDefaultAvatar() {
    if (authorAvatar == null || getContext() == null) return;

    // 确保头像显示为圆形
    GradientDrawable circleDrawable = new GradientDrawable();
    circleDrawable.setShape(GradientDrawable.OVAL);
    circleDrawable.setColor(Color.TRANSPARENT);
    circleDrawable.setStroke(1, Color.TRANSPARENT);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
        authorAvatar.setBackground(circleDrawable);
    } else {
        authorAvatar.setBackgroundDrawable(circleDrawable);
    }

    // 使用RoundedBitmapDrawable确保图片也是圆形的
    Drawable drawable = ContextCompat.getDrawable(getContext(),
R.drawable.user_avatar);
    if (drawable instanceof BitmapDrawable) {
        Bitmap bitmap = ((BitmapDrawable) drawable).getBitmap();
        RoundedBitmapDrawable roundedBitmapDrawable =
RoundedBitmapDrawableFactory.create(getResources(), bitmap);
        roundedBitmapDrawable.setCircular(true);
        authorAvatar.setImageDrawable(roundedBitmapDrawable);
    } else {
        authorAvatar.setImageResource(R.drawable.user_avatar);
    }

    // 添加额外的null检查
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        authorAvatar.setClipToOutline(true);
    }
}

```

1. 头像显示修复：
   - 使用Glide加载Post对象中Author的avatar URL
   - 添加了错误处理和占位图显示
   - 确保头像显示为圆形
2. 图片显示和比例控制：
   - 修改ImagePagerAdapter以直接使用Post.Clip对象
   - 根据首图计算宽高比，并限制在3:4 ~ 16:9之间
   - 动态调整图片容器高度以匹配计算的比例
   - 非首图使用centerCrop确保内容充满容器
   - 使用Glide加载网络图片，支持占位图和错误处理
3. 其他优化：
   - 显示正确的点赞数
   - 优化了图片加载逻辑，添加了异常处理
   - 移除了对PostDetail类的依赖，直接使用Post对象

详情页没有正确显示图片和正文，是因为：

在PostDetailFragment中，`postDetail`变量始终为null，而`setupData()`方法开头有一个检查：

代码块

```
1  private void setupData() {
2      if (postDetail == null) return;
3      // 后续设置内容的代码...
4  }
```

导致当`postDetail`为null时，方法直接返回，没有执行后续的内容设置。我们需要修改`setupData()`方法，让它直接使用从`getArguments()`获取的Post对象，而不是依赖于`postDetail`变量。以下是修复方案：

代码块

```
1  private void setupData() {
2      Post post = (Post) getArguments().getSerializable(ARG_POST);
3      if (post == null) return;
4
5      // 设置作者信息
6      authorName.setText(post.getAuthorName());
```

```
        // 设置作者头像
        loadAuthorAvatar();

        // 检查是否已关注该作者
        String authorId = post.getAuthor() != null ? post.getAuthor().getUser_id()
    : "unknown_author";
        SharedPreferences prefs =
    getActivity().getSharedPreferences(PREF_FOLLOWED_AUTHORS,
    Context.MODE_PRIVATE);
        isFollowing = prefs.getBoolean(authorId, false);
        updateFollowButton();

        // 设置图片适配器
        setupImageAdapter();

        // 设置帖子内容
        postTitle.setText(post.getTitle());
        postContent.setText(post.getContent() != null ? post.getContent() : "暂无内
    容");

        // 设置话题标签
        // 简单实现话题标签显示
        topicContainer.removeAllViews();
        if (post.getHashtag() != null && !post.getHashtag().isEmpty()) {
            TextView topicView = (TextView) LayoutInflater.from(getContext())
                    .inflate(R.layout.item_topic, topicContainer, false);
            topicView.setText("#" + post.getHashtag());
            topicView.setOnClickListener(v -> handleTopicClick(post.getHashtag()));
            topicContainer.addView(topicView);
        } else {
            TextView topicView = (TextView) LayoutInflater.from(getContext())
                    .inflate(R.layout.item_topic, topicContainer, false);
            topicView.setText("#生活");
            topicView.setOnClickListener(v -> handleTopicClick("生活"));
            topicContainer.addView(topicView);
        }

        // 设置发布日期
        postDate.setText(formatDate(post.getCreate_time()));

        // 设置点赞状态和数量
        updateLikeStatus(isLiked);
        likeCount.setText(String.valueOf(currentLikeCount));
    }
```

显示如下所示：

解决问题：详情页面中，帖子的创建时间，日期显示不正确，都是显示1.21号，

实际上就是formatDate写的不对，没有判断时间是秒还是毫秒，

代码块

```
1   private String formatDate(long timestamp) {
2       Log.d("time", "原始timestamp: " + timestamp);
3
4       // 检查时间戳是否可能是秒级别的（如果值太小）
5       // 秒级时间戳通常在10^9左右，毫秒级在10^12左右
6       if (timestamp < 10000000000L) {
7           Log.d("time", "检测到可能的秒级时间戳，转换为毫秒");
8           timestamp = timestamp * 1000; // 将秒转换为毫秒
9       }
10
```

```
11      Log.d("time", "处理后的timestamp: " + timestamp);
12
13      long now = System.currentTimeMillis();
14      long diff = now - timestamp;
15      Log.d("time", "当前时间: " + now);
16      Log.d("time", "时间差: " + diff);
17
18      // 24小时内
19      if (diff < 24 * 60 * 60 * 1000) {
20          // 获取当前日期的起始时间（今天0点）
21          Calendar today = Calendar.getInstance();
22          today.set(Calendar.HOUR_OF_DAY, 0);
23          today.set(Calendar.MINUTE, 0);
24          today.set(Calendar.SECOND, 0);
25          today.set(Calendar.MILLISECOND, 0);
26          long todayStartTime = today.getTimeInMillis();
27
28          // 获取昨天的起始时间
29          Calendar yesterday = Calendar.getInstance();
30          yesterday.add(Calendar.DAY_OF_MONTH, -1);
31          yesterday.set(Calendar.HOUR_OF_DAY, 0);
32          yesterday.set(Calendar.MINUTE, 0);
33          yesterday.set(Calendar.SECOND, 0);
34          yesterday.set(Calendar.MILLISECOND, 0);
35          long yesterdayStartTime = yesterday.getTimeInMillis();
36
37          if (timestamp >= todayStartTime) {
38              // 今天
39              String result = new SimpleDateFormat("HH:mm",
    Locale.getDefault()).format(new Date(timestamp));
40              Log.d("time", "返回今天格式: " + result);
41              return result;
42          } else if (timestamp >= yesterdayStartTime) {
43              // 昨天
44              String result = "昨天 " + new SimpleDateFormat("HH:mm",
    Locale.getDefault()).format(new Date(timestamp));
45              Log.d("time", "返回昨天格式: " + result);
46              return result;
47          }
48      }
49      // 7天内
50      else if (diff < 7 * 24 * 60 * 60 * 1000) {
51          long days = diff / (24 * 60 * 60 * 1000);
52          String result = days + "天前";
53          Log.d("time", "返回几天前格式: " + result);
54          return result;
55      }
```

```
56        // 其他情况
57        else {
58            String result = new SimpleDateFormat("MM-dd",
    Locale.getDefault()).format(new Date(timestamp));
59            Log.d("time", "返回日期格式: " + result);
60            return result;
61        }
62        // 默认情况（应该不会执行到这里）
63        String result = new SimpleDateFormat("MM-dd",
    Locale.getDefault()).format(new Date(timestamp));
64        Log.d("time", "返回默认日期格式: " + result);
65        return result;
66    }
```

解决问题：点击详情页，但是回退到首页，又会重新刷新。

原因是当从详情页返回首页时，HomeFragment的onCreateView()方法会被重新调用，导致fetchDataFromApi()再次执行，重新加载数据而不是显示之前的数据

解决方案：将数据加载逻辑从onCreateView()移到onCreate()方法中，这样数据只会在Fragment首次创建时加载一次，从详情页返回时不会重新加载

代码块

```
1   @Override
2   public void onCreate(Bundle savedInstanceState) {
3       super.onCreate(savedInstanceState);
4       // 初始化数据列表
5       posts = new ArrayList<>();
6       // 在Fragement创建时获取数据，而不是每次创建视图时获取
7       fetchDataFromApi();
8   }
```

实现每个帖子的点赞状态、关注状态的 本地磁盘持久化存储

实现思路：

1. 点赞状态的本地持久化存储 - 使用SharedPreferences保存每个帖子ID对应的点赞状态
2. 关注状态的本地持久化存储 - 使用SharedPreferences保存每个作者ID对应的关注状态
3. 在加载数据时自动应用本地存储的状态
4. 修复PostDetailFragment中使用postDetail导致的空指针问题

修改toggleFollowStatus、toggleLikeStatus这两个函数，并设置setupData的逻辑，

然后修改HomeFragment，实现点赞状态的持久化

同时注意完善，首页展示与详情页展示中，根据本地的点赞状态同步点赞数，实现首页与详情的点赞显示一致。

设计下拉刷新、上滑加载更多和空态页面

修改布局文件，添加下拉控件和空态页面。

修改HomeFragment.java实现下拉刷新和加载更多

下拉刷新用到了swipeRefreshLayout。

下拉刷新的逻辑：

- 在 `fragment_home.xml` 中使用控件包裹RecycleView
- 在 `HomeFragment.java` 中设置刷新监听器
- 在ViewModel层中进行处理，在 `PostViewModel` 中的refreshData()方法，检查加载状态，调用repository层的fetchPosts获取数据，然后在回调中处理成功或失败数据。
- 成功获取数据后，清楚旧数据，添加新数据，并应用本地点赞状态
- 通过LiveData通知UI更新

上滑不断加载新的内容的实现。

下拉刷新并不需要缓存，而是直接刷新。

虽然实现了不断上滑刷新，但是点击详情回退，又出现了刷新这一操作。解决这一问题：

1. 添加状态保存变量：
    - `firstLoad`：标记是否是首次加载，确保只有在第一次创建Fragment时才加载数据
    - `scrollPosition`：保存滚动位置的索引
    - `scrollOffsets`：保存滚动的偏移量，用于更精确地恢复位置
2. 修改滚动监听：
    - 添加 `onScrollStateChanged` 方法，在用户停止滚动时保存当前位置
    - 记录第一个可见item的位置和偏移量
3. 优化onResume方法：
    - 移除原来的完整数据加载逻辑
    - 只应用本地存储的点赞状态更新
    - 通知适配器数据已更改，但不会重新加载整个列表
    - 恢复用户之前浏览的滚动位置

4. 保留下拉刷新功能：
    - 下拉刷新时仍然会重新加载数据
    - 刷新时重置保存的位置信息

具体实现方式是在HomeFragment的setupRecyclerView，实现触发逻辑，当满足触发条件调用PostViewModel中的数据加载逻辑loadMore方法。

注意区分loadmore与refreshData方法，（在app优化首次加载中实现的）

refreshData方法在这个过程中实现了一个cache保存，先尝试保存到cache，在refreshData中

功能定位：

refreshData()：

- 用于首次加载和下拉刷新场景
- 从 `HomeFragment.java` 可以看到，它在Fragment的 `onCreate` 方法（首次加载）和下拉刷新监听器中被调用
- 功能是替换现有数据，获取最新的第一页数据

loadMoreData()：

- 用于上滑加载更多场景
- 从 `HomeFragment.java` 可以看到，它在RecyclerView滚动到底部时被调用
- 功能是追加新数据到现有列表末尾

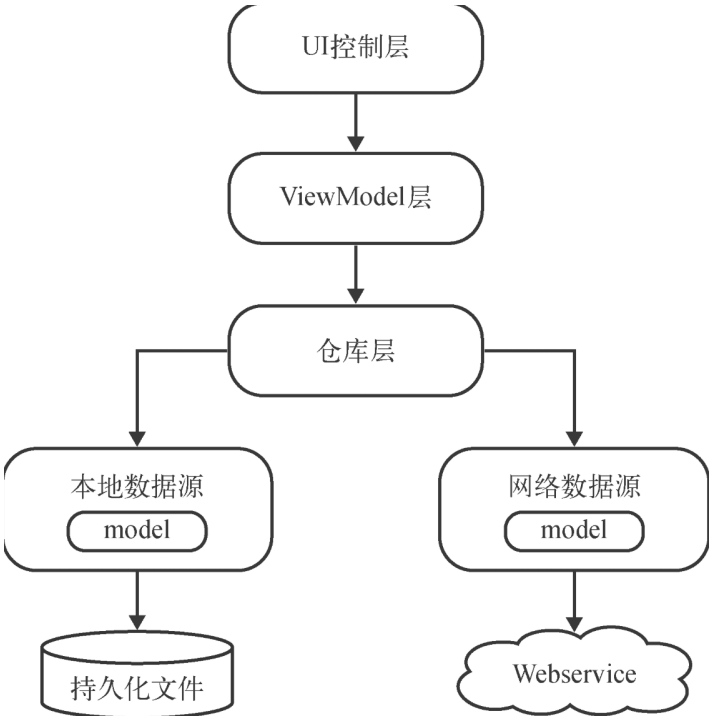
在现有代码基础上，重构框架，实现MV VM架构

参考《第一行代码》中的15.3搭建MVVM项目架构

MVVM(Model-View-ViewModel)是一种高级项目架构模式，

- Model是数据模型部分
- View是界面展示部分
- ViewModel，理解为一个连接数据模型和界面展示的桥梁，从而实现让业务逻辑和界面展示分离的程序结构设计

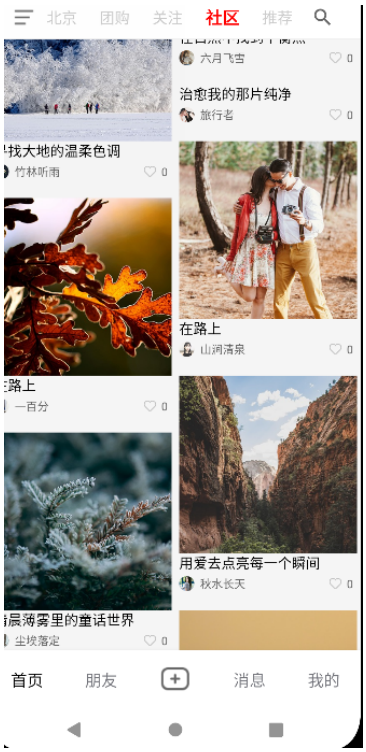
当然，一个优秀的项目架构除了包含上述3部分内容，还应该包含仓库、数据源等，示例MVVM项目架构图如下所示：



在我的项目代码中，重构过程如下：

1. MVVM架构分层：
    - Model层：`Post`模型类负责数据结构
    - View层：`HomeFragment`和布局文件负责UI展示
    - ViewModel层：`PostViewModel`负责业务逻辑和数据管理
    - Repository层：`PostRepository`负责数据获取和存储
2. 主要改进：
    - 使用LiveData进行数据观察和UI更新
    - 将数据获取和处理逻辑从Fragment移至Repository
    - 将业务逻辑从Fragment移至ViewModel
    - 将适配器提取为独立类，减少Fragment的复杂度
    - 通过接口实现Fragment和Adapter之间的通信

完成情况如下：能够正常展示，并刷新。



分析了PostDetailFragment的完整代码，可以看到它目前存在以下问题：

- 代码职责过多，既处理UI展示又处理业务逻辑
- 数据操作和UI耦合在一起
- 本地存储操作直接在Fragment中进行
- 缺少生命周期安全的数据管理

优化如下：

1. 关注点分离：
    - Fragment负责UI展示和用户交互
    - ViewModel负责业务逻辑和数据管理
    - Repository负责数据获取（未来可扩展）
2. 生命周期安全：
    - 使用LiveData和ViewModel确保配置变更时数据不丢失
    - 使用getViewLifecycleOwner()确保观察者在Fragment生命周期内有效
3. 可测试性提升：
    - ViewModel中的业务逻辑可以独立测试
    - Fragment的UI展示逻辑更简单，更容易验证
4. 代码可维护性：
    - 每个类职责单一，更容易理解和修改
    - 减少了Fragment中的代码量，提高了可读性

解决问题：为了完全移除上拉加载时的动画显示，我们需要修改PostViewModel中的loadMoreData方法，让它在加载更多数据时不设置loading状态。同时，我们还需要修改HomeFragment.java，确保即使在初始加载时也不显示loading动画：

完善滚动位置保存和恢复

1. 添加了保存滚动偏移量的变量，使位置恢复更精确
2. 在滚动停止时保存更完整的滚动位置信息，包括每个列的偏移量
3. 在从详情页返回时（onResume）使用更精确的方法恢复滚动位置
4. 确保在下拉刷新时正确重置保存的位置信息

进一步的优化，

1. **即时保存位置**：在用户点击卡片跳转前，立即调用 `saveCurrentScrollPosition()` 方法保存当前滚动状态，而不是等待滚动停止或依赖其他时机
2. 清理重复逻辑：移除了onResume方法中重复的滚动恢复代码，避免冲突
3. 精确恢复：使用 `scrollToPositionWithOffset` 方法，结合保存的位置和偏移量，实现更精确的滚动位置恢复
4. 专门的保存方法：将位置保存逻辑封装到单独的方法中，使代码更清晰

增加导航栏"我的"

首先创建 `MineViewModel` 类来管理用户个人信息的数据

接下来创建 `MineFragment` 类来处理UI展示和用户交互：

创建 `fragment_mine.xml` 布局文件来定义页面的UI结构：

最后，修改 `MainActivity.java` 中的 `setTabClickListener` 方法，添加“我的”按钮的点击事件：

初始效果如下：

完善底部导航栏，点击不同栏，实现颜色切换，

完善保留首页状态，当从我的跳转到首页，不刷新还是之前的状态。如下设置：

1. 通过成员变量保存HomeFragment和MineFragment的实例引用
2. 当需要显示某个Fragment时，首先检查该实例是否已存在：
    - 如果不存在，则创建新实例并添加到容器中
    - 如果已存在，则直接显示，不再重新创建
3. 使用show/hide方法管理Fragment的可见性，而不是replace方法
4. 维护currentFragment变量跟踪当前显示的Fragment，避免不必要的操作

这样，当你从首页切换到"我的"页面，再切回首页时，页面将保持之前的状态，不会重新加载或刷新。Fragment的生命周期方法（如onCreateView）只会在首次创建时调用一次，切换时不会重复执行，从而实现了页面状态的保持。

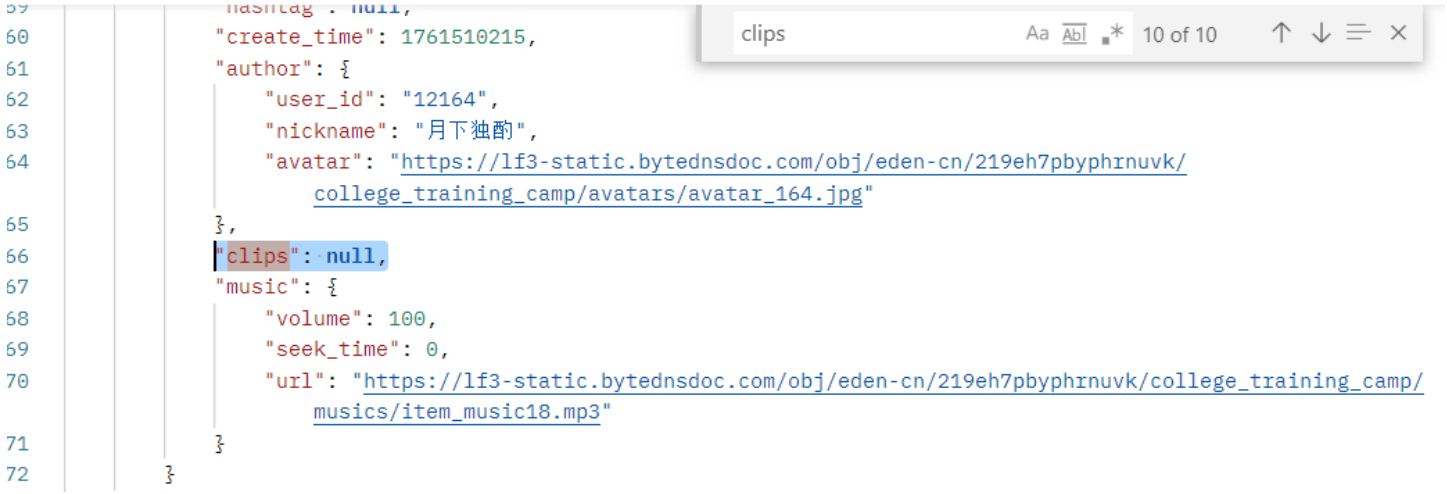
修复我的和首页页面切换逻辑：

问题的主要原因是混合使用了 `replace` 和 `show/hide` 两种Fragment切换方式，导致Fragment被不必要地重新创建。通过统一使用 `show/hide` 方式，并只在Fragment第一次创建时加载数据，可以解决首页数据刷新的问题。

修改后，从首页切换到"我的"页面再切换回来时，首页将保持原有的数据和滚动位置，不会重新加载数据。

解决clips:null的情况

存在没有图片的帖子，接口返回数据如下：



在循环解析每个帖子的开始部分，添加了对clips字段是否为null的检查。当遇到clips字段为null的帖子时，会直接跳过后续的解析过程，不将该帖子添加到postList中，从而实现了不展示这些帖子的要求。同时添加了日志记录，方便调试和查看跳过的帖子信息。

进入app优化首次加载

优化方式：在现有代码基础上进行增强，以提升首次启动的响应速度。

实现本地缓存

在 `PostRepository.java` 中添加保存缓存，从缓存中读取数据两个方法。

在 `PostViewModel` 中修改refreshData方法，先加载缓存数据，同时异步请求新数据，并将数据更新缓存。

优化HomeFragment初始化流程

在onCreate时就加载数据，而不是等到onCreateView时加载数据。

延迟加载非关键资源

修改 `HomeFragment` 中的 `setupRecyclerView` ，延迟加载刷新监听

优化RecyclerView性能

在PostAdapter中进行优化，

优化启动白屏界面

我们可以通过实现启动主题(Launcher Theme)来优化。这种方法可以在应用加载过程中显示品牌Logo或启动画面，而不是空白屏幕

定义好主题，为MainActivity设置启动主题，在 `androidmanifest.xml` 中设置。

页面转场动画实现

修改PostAdapter：为卡片图片设置唯一的过渡名称

修改HomeFragment：在启动详情页时设置共享元素转场

修改PostDetailActivity：设置详情页图片的过渡名称

修改ImagePagerAdapter：为ViewPager中的图片设置过渡名称

创建自定义转场动画：实现图片放大缩小和背景渐显渐隐效果

进入详情页的放大动画不明显主要是因为**共享元素设置不匹配**和**动画配置不够明显**

图片进度条间隔

首先修改fragment_post_detail.xml文件中的进度指示器，

然后修改PostDetailFragment.java文件

优化首页界面中展示出的图片缓存，实现详情加载

1. MyApplication.java 实现错误：它被错误地实现为 `AppGlideModule` 而非标准的 Android `Application` 类，且未在 AndroidManifest.xml 中注册，导致缓存配置完全未生效。
2. 过度预加载：当前的预加载策略会预加载所有帖子的所有图片，这会严重影响首页加载性能。
3. 缓存机制配置不当：虽然代码中有缓存配置，但由于上述问题，实际上并未生效。

解决详情页是单个Activity多Fragment的问题

现在逻辑是ImageViewHolder的bind方法中，实现替换首页的Fragment来展示详情页的，

应该使用单独的一个Activity，创建新的PostDetailActivity，并且创建ImagePostAdapter

FragmentPostDetailBinding是Android的View Binding功能生成的类，它会自动根据fragment_post_detail.xml布局文件生成对应的绑定类。如果你的项目启用了View Binding，Android Studio会自动为你生成这个类。你可以在模块的build.gradle文件中添加以下配置来启用View Binding：

然后设置和PostDetailFagment类似的功能，实现页面详情的展示。

这样设置过后打开详情页就快多了

解决缓存数据的时候

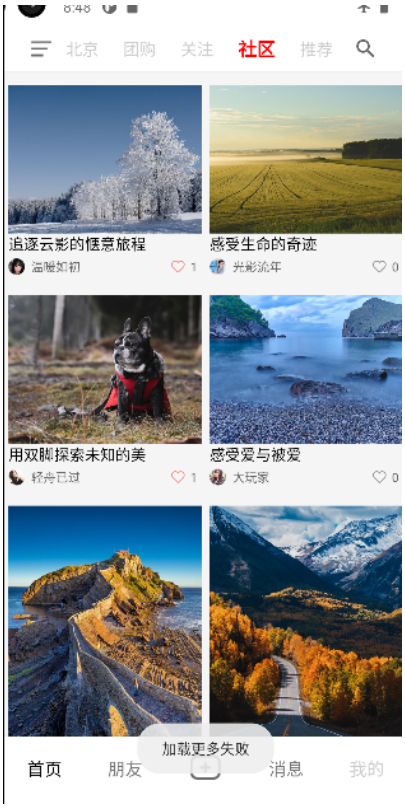
在下拉刷新流程中用到了refreshData，这里并不需要设置缓存，而是直接刷新替换。

缓存使用，因该在测试无网环境下，获取不到网络请求数据，读取缓存中的数据。

在postviewmodel中fetchPosts方法中，如果onsuccess，则将数据保存到cache中，如果onFaliure，则先尝试缓存数据，读不到再尝试使用测试数据。

```
代码块
1  @Override
2              public void onFailure(String error) {
3                  Log.e(TAG, "刷新失败: " + error);
4
5                  // 加载失败时显示测试数据
6                  posts.clear();
7  //                  List<Post> testPosts = repository.generateTestPosts(10);
8                  List<Post> cachedPosts = repository.readPostsFromCache();
9                  if (cachedPosts != null && !cachedPosts.isEmpty()) {
10                     // 应用本地存储的点赞状态
11                     repository.applyLocalLikeStatus(cachedPosts);
12
13                     posts.clear();
14                     posts.addAll(cachedPosts);
15                     postsLiveData.setValue(new ArrayList<>(posts));
16
17                     hasMoreData = true;
18                     hasMoreDataLiveData.setValue(true);
19
20                     isLoading = false;
21                     isLoadingLiveData.setValue(false);
22                     isRefreshingLiveData.setValue(false);
23                     isErrorLiveData.setValue(false); // 不显示错误状态，因为使用了
   缓存数据
24                 } else {
25                     // 如果没有缓存，才显示测试数据
26                     posts.clear();
27                     List<Post> testPosts = repository.generateTestPosts(10);
28                     posts.addAll(testPosts);
29                     postsLiveData.setValue(new ArrayList<>(posts));
30
31                     hasMoreData = true;
32                     hasMoreDataLiveData.setValue(true);
33
34                     isLoading = false;
35                     isLoadingLiveData.setValue(false);
36                     isRefreshingLiveData.setValue(false);
37                     isErrorLiveData.setValue(false); // 不显示错误状态，因为使用了
   测试数据
38                 }
39             }
```

这样就实现了飞行模式下，加载兜底数据，先加载缓存数据，没有缓存再加载生成的测试数据。



修改关注状态按钮颜色

在 `res/drawable/` 目录下创建 `button_follow_border.xml`，

在 `post_fragemnt_detail.xml` 中应用该资源，

应用布局按钮显示为紫色而不是红色边框，通常有以下几个原因：

Android 12+ 使用了 Material Design 3，默认会给按钮添加紫色主题色

- 方法A：清除默认的背景着色
- 使用 AppCompatButton，这个方法成功了

代码块

```
<androidx.appcompat.widget.AppCompatButton
    android:id="@+id/follow_button"
    android:layout_width="64dp"
    android:layout_height="38dp"
    android:layout_marginEnd="16dp"
    android:gravity="center"
    android:includeFontPadding="false"
    android:minWidth="60dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:text="关注"
    android:textColor="#fe2c55"
    android:textSize="14sp"
    android:background="@drawable/button_follow_border"
    android:backgroundTint="@null"
```

```
16        app:layout_constraintBottom_toBottomOf="parent"
17        app:layout_constraintEnd_toEndOf="parent"
18        app:layout_constraintTop_toTopOf="parent"
19        />
```

这里进一步优化为：创建不同状态的边框（关注/已关注）

解决刷新过程中图片变换的效果

添加背景音乐

添加ExoPlayer依赖，然后创建音乐播放管理器，

然后修改详情页布局，添加静音按钮，并创建需要的xml布局文件。

然后修改详情页activity，集成音乐播放功能。

修改Application类，处理冷启动重置。

1. 静音按钮：在图片容器右下角显示，点击可切换静音状态
2. 播控逻辑：
   - 打开详情页时根据记忆状态播放/静音
   - 关闭页面、打开其他页面、退后台时暂停播放
   - 返回详情页、回前台时恢复播放
3. 状态记忆：
   - APP生命周期内保持静音状态
   - 打开不同作品详情页继承上次的静音状态
   - APP冷启动后重置为非静音状态

使用了Google推荐的ExoPlayer，性能稳定且功能完善

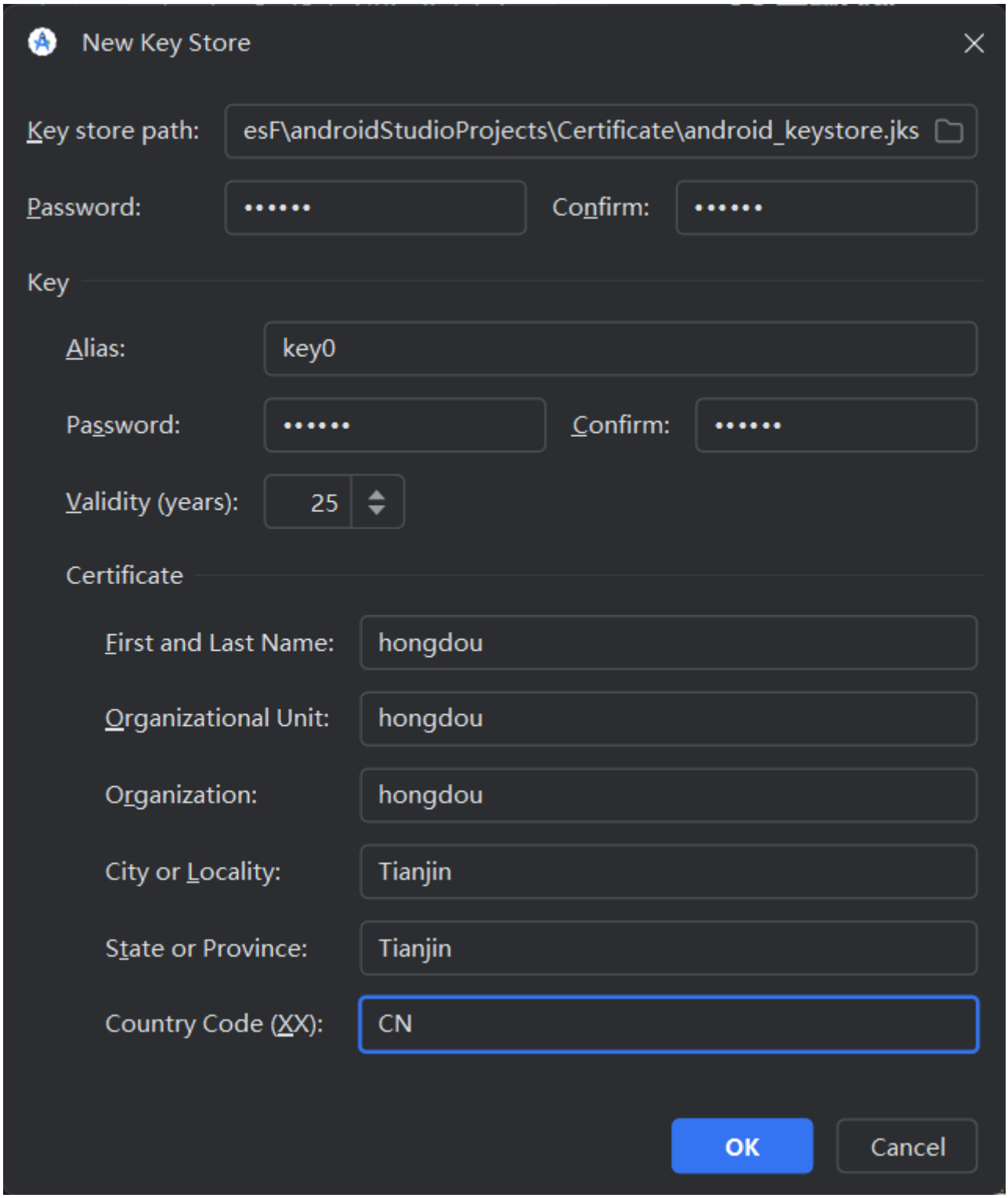
音乐图标按钮的位置：

代码块

```
1   <RelativeLayout
2       android:id="@+id/image_container"
3       android:layout_width="match_parent"
4       android:layout_height="wrap_content">
5
6       <!-- 图片ViewPager -->
7       <androidx.viewpager2.widget.ViewPager2
8           android:id="@+id/image_view_pager"
9           android:layout_width="match_parent"
10          android:layout_height="wrap_content" />
```

```
11
12      <!-- 底部遮罩 + 进度条 -->
13      <LinearLayout
14          android:id="@+id/bottom_bar"
15          android:layout_width="match_parent"
16          android:layout_height="wrap_content"
17          android:layout_alignBottom="@id/image_view_pager"
18          android:background="#80000000"
19          android:orientation="vertical"
20          android:padding="8dp">
21
22          <LinearLayout
23              android:id="@+id/progress_dots_container"
24              android:layout_width="match_parent"
25              android:layout_height="8dp"
26              android:layout_marginTop="4dp"
27              android:gravity="center"
28              android:orientation="horizontal" />
29      </LinearLayout>
30
31      <!-- 将按钮放到最外层 RelativeLayout，并对齐到底部bar的右上角 -->
32      <ImageView
33          android:id="@+id/mute_button"
34          android:layout_width="40dp"
35          android:layout_height="40dp"
36          android:layout_alignTop="@id/bottom_bar"
37          android:layout_alignEnd="@id/bottom_bar"
38          android:layout_marginTop="-20dp"    <!-- 可调，使其悬浮在bar之上 -->
39          android:layout_marginEnd="8dp"
40          android:background="@drawable/circle_background"
41          android:padding="8dp"
42          android:src="@drawable/ic_volume_up"
43          android:tint="@color/white" />
44  </RelativeLayout>
45
```

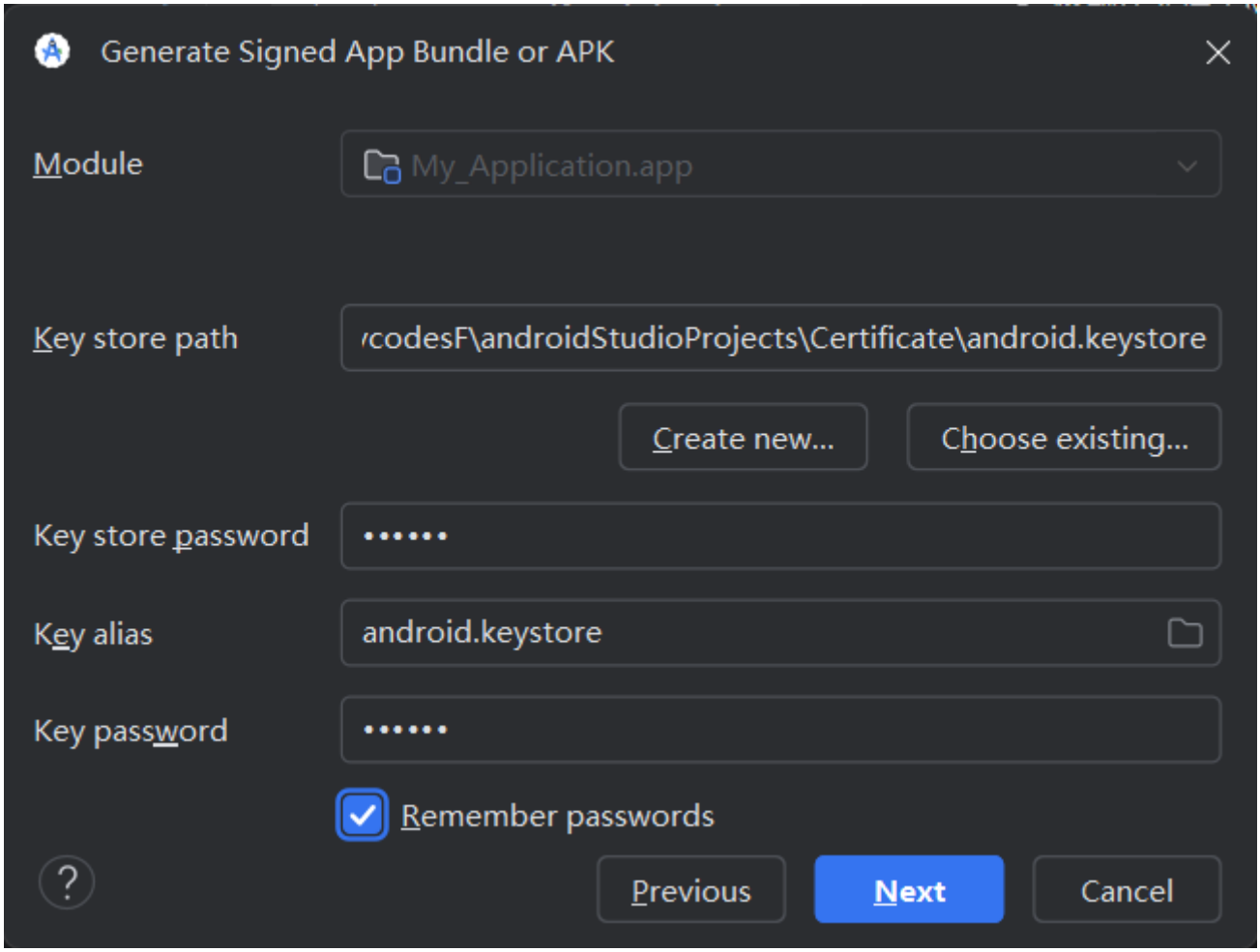
打包为APK



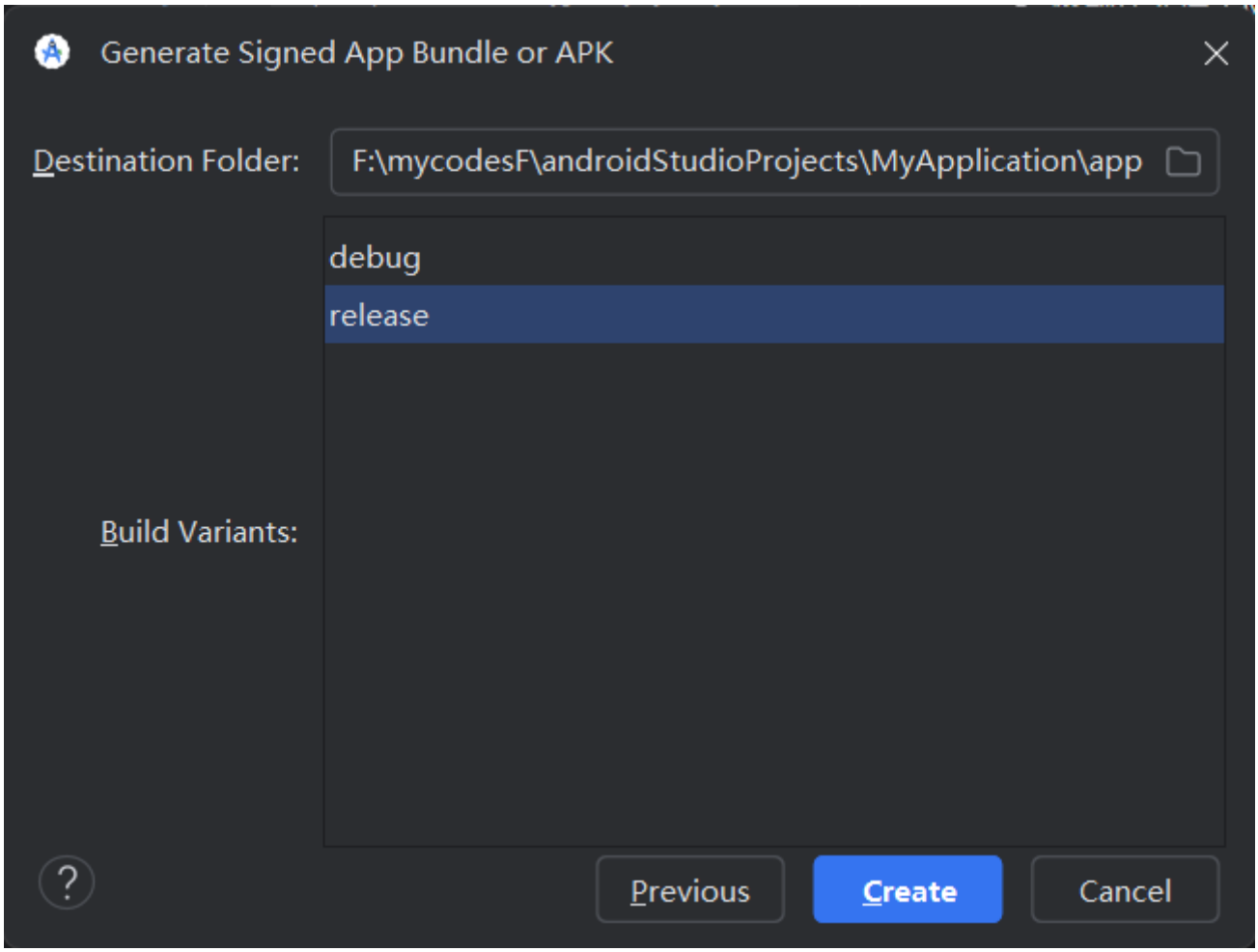
如果上述在android sutdio中不能创建成功，则在cmd中实现。

```
F:\mycodesF\androidStudioProjects\Certificate>keytool -genkey -alias android.keystore -keyalg RSA -validity 36500 -keyst
ore android.keystore
输入密钥库口令:

密钥库口令太短 - 至少必须为 6 个字符
输入密钥库口令:

再次输入新口令:

您的名字与姓氏是什么?
  [Unknown]:  Hongdou
您的组织单位名称是什么?
  [Unknown]:  Hongdou
您的组织名称是什么?
  [Unknown]:  Hongdou
您所在的城市或区域名称是什么?
  [Unknown]:  Tianjin
您所在的省/市/自治区名称是什么?
  [Unknown]:  Tianjin
该单位的双字母国家/地区代码是什么?
  [Unknown]:  CN
CN=Hongdou, OU=Hongdou, O=Hongdou, L=Tianjin, ST=Tianjin, C=CN是否正确?
  [否]:  y

正在为以下对象生成 2,048 位RSA密钥对和自签名证书 (SHA256withRSA) (有效期为 36,500 天):
        CN=Hongdou, OU=Hongdou, O=Hongdou, L=Tianjin, ST=Tianjin, C=CN
```

密码为123456



选择release版本，



apk存放路径：

- 切换左上角的 `Android` 为 `Project`，此时可以发现app项目下多了个release文件夹，文件夹下有apk文件
- 点击该apk文件，右键 `Open In` -> `Explorer`，就能打开到该apk所在的文件夹

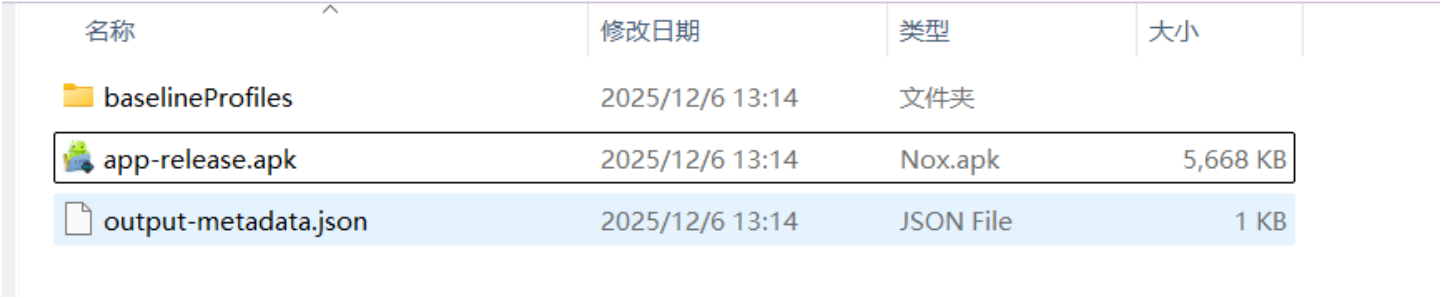
注意打包不能完成的，需要配置一下镜像源或者查看梯子

代码块

```
// settings.gradle.kts
pluginManagement {
    repositories {
        // 阿里云镜像（插件）
        maven { url = uri("https://maven.aliyun.com/repository/gradle-plugin")
}
        maven { url = uri("https://maven.aliyun.com/repository/public") }

        // 华为镜像
        maven { url = uri("https://repo.huaweicloud.com/repository/maven/") }

        // 腾讯云镜像
        maven { url =
uri("https://mirrors.cloud.tencent.com/nexus/repository/maven-public/") }

```

```
        // 官方源（最后）
        gradlePluginPortal()
        google()
        mavenCentral()
    }
}

dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.PREFER_SETTINGS)
    repositories {
        // Google Android 仓库镜像（关键）
        maven {
            name = "Google"
            url = uri("https://maven.aliyun.com/repository/google")
            content {
                includeGroupByRegex("com\\.android.*")
                includeGroupByRegex("com\\.google.*")
                includeGroupByRegex("androidx.*")
            }
        }

        // Google 官方（备用）
        maven {
            name = "GoogleOfficial"
            url = uri("https://dl.google.com/dl/android/maven2/")
            content {
                includeGroupByRegex("com\\.android.*")
                includeGroupByRegex("com\\.google.*")
            }
        }

        // 中央仓库镜像
        maven {
            name = "AliyunCentral"
            url = uri("https://maven.aliyun.com/repository/central")
        }

        mavenCentral()
    }
}
```

然后下载夜神模拟器，将打包好的apk拖到模拟器中即可使用。



名称	修改日期	类型	大小
baselineProfiles	2025/12/6 13:14	文件夹	
app-release.apk	2025/12/6 13:14	Nox.apk	5,668 KB
output-metadata.json	2025/12/6 13:14	JSON File	1 KB

个人思考

总结

在理论层面，通过系统性的课程学习，我构建了完整的客户端技术知识体系。不仅掌握了Android开发的四大核心组件、Jetpack架构组件（如ViewModel、LiveData、Navigation）等关键技术，更关键的是理解了客户端开发的设计思想与最佳实践，例如如何通过合理的应用架构实现关注点分离，如何运用“负负得正”等算法思想优化逻辑，以及现代开发中单Activity多Fragment的设计模式优势。

在实践层面，通过完成课题项目并在导师的悉心指导下反复迭代，我将理论知识转化为解决实际问题的能力。从页面布局的实现、组件间通信，到数据层的设计、缓存策略的应用，我完整地体验了客户端功能从设计、编码到调试、优化的全流程。这一过程不仅让我熟练了工具和技术，更重要的是培养了我对代码结构、性能考量与用户体验的敏感性，学会了如何编写更健壮、更易维护的代码。

当然这个过程中也有许多不理解，需要自己动手实践的地方，还有就是对于某个需求，要充分结合现有知识，以及AI工具来更好的完成该功能，当自己不能完成的时候，需要和导师以及身边人询问并交流，不能自己一个人琢磨。

总而言之，本次训练营对我而言是一次宝贵的“学以致用”之旅。它既拓宽了我的技术视野，明确了客户端工程师的成长路径，也通过项目实战夯实了我的工程能力，为今后从事专业的客户端开发工作奠定了坚实的基础

改进建议

架构优化方向

1. 引入依赖注入：使用Dagger或Hilt管理依赖
2. 状态管理优化：考虑使用StateFlow替代部分LiveData
3. 导航组件：使用Navigation Component管理页面跳转
4. 数据持久化：引入Room数据库进行本地数据管理

性能优化

- 图片加载优化（Glide/Picasso）
- 列表项复用优化
- 内存泄漏检测和预防