RedCourse

框架设计

基于 **Navigation** 组件的单 Activity 安卓应用框架。

一、应用概述

我们设计了一个 `MainActivity` 的应用，它是应用内容展示的入口 Activity。它使用了以下几个关键组件：

- `View Binding`：一种安全地访问布局文件中 View 的机制，替代 `findViewById`。
- `Navigation Component`：用于管理应用内导航的框架，简化了 Fragment 之间的跳转和数据传递。

二、构建步骤

1. 设置布局 (ActivityMainBinding)

首先使用 `ActivityMainBinding` 来加载并绑定布局文件 `activity_main.xml`。

关键代码：

```java
binding = ActivityMainBinding.inflate(getLayoutInflater());
setContentView(binding.getRoot());
```

- `ActivityMainBinding.inflate(getLayoutInflater())`：这行代码会解析 `activity_main.xml` 布局文件，并创建一个 `ActivityMainBinding` 实例。这个实例包含了布局文件中所有具有 ID 的 View 的引用。
- `setContentView(binding.getRoot())`：将 `binding.getRoot()`（即布局文件的根 View）设置为 Activity 的内容视图，这样我们的布局就会显示在屏幕上。

2. 初始化 Navigation 组件

接下来，我们初始化 Navigation 组件，并将其与底部导航栏关联起来。

关键代码：

```java
NavController navController = Navigation.findNavController(this, R.id.
nav_host_fragment_activity_main);
NavigationUI.setupWithNavController(binding.navView, navController);
```

说明：

- `Navigation.findNavController(this, R.id.nav_host_fragment_activity_main)`：这行代码查找并获取了 `NavController` 的实例。`nav_host_fragment_activity_main` 是 `activity_main.xml` 中 `NavHostFragment` 的 ID，它是承载所有 Fragment 的容器。
- `NavigationUI.setupWithNavController(binding.navView, navController)`：这行代码将 `NavController` 与 `BottomNavigationView`（`binding.navView`）关联起来。这样，当用户点击底部导航栏的不同选项时，`NavController` 会自动切换到对应的 Fragment。

三、`activity_main.xml` 布局文件（关键部分）

`activity_main.xml` 布局文件至关重要，它包含以下关键组件：

XML/HTML

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/nav_view"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="0dp"
        android:layout_marginEnd="0dp"
        android:background="?android:attr/windowBackground"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:menu="@menu/bottom_nav_menu" />

    <fragment
        android:id="@+id/nav_host_fragment_activity_main"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toTopOf="@id/nav_view"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/mobile_navigation" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

说明：

- `BottomNavigationView`：底部导航栏，用于在不同的顶级目的地之间切换。

- `fragment`: 作为 `NavHostFragment` 的容器，它负责承载和管理 Fragment。
  - `android:name="androidx.navigation.fragment.NavHostFragment"`：指定这个 View 是一个 `NavHostFragment`。
  - `app:defaultNavHost="true"`：表示这个 `NavHostFragment` 会处理系统的返回按钮事件。
  - `app:navGraph="@navigation/mobile_navigation"`：指定导航图资源文件，这个文件定义了应用中的导航结构和 Fragment 之间的关系。

四、导航图 (mobile_navigation.xml)

`mobile_navigation.xml` 文件 (在 `navigation` 资源文件夹中创建) 定义了应用的导航图。它描述了应用中的各个目的地（通常是 Fragment）以及它们之间的导航关系。您需要在这个文件中定义每个 Fragment 以及它们之间的跳转动作 (Action)。

组件模块

数据库设计

1. 数据库概览

该应用使用 SQLite 数据库来存储数据，并利用 SQLiteOpenHelper 类来管理数据库的创建和版本升级。

数据库名称 : AppDatabase

数据库版本 : 1

2. 表结构设计

数据库包含四个表：

users (用户信息表): 存储用户的基本信息。

watch_history (观看历史表): 记录用户的视频观看历史。

favorites (收藏表): 记录用户收藏的视频。

关键代码 (CREATE_TABLE_USERS):

```java
private static final String CREATE_TABLE_USERS = "CREATE TABLE " + TABLE_USERS + "("
        + KEY_USER_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
        + KEY_USERNAME + " TEXT NOT NULL,"
        + KEY_PASSWORD + " TEXT NOT NULL,"
        + KEY_REGISTER_DATE + " DATETIME DEFAULT CURRENT_TIMESTAMP,"
        + KEY_PHONE + " TEXT UNIQUE)";
```

这段代码定义了创建 users 表的 SQL 语句。其中：

PRIMARY KEY AUTOINCREMENT 表示 id 列为主键，且自动递增。

NOT NULL 约束确保 username 和 password 列不能为空。

DEFAULT CURRENT_TIMESTAMP 表示 register_date 列的默认值为当前时间戳。

UNIQUE 约束确保 phone 列的值是唯一的。

2.2 watch_history 表

列名	数据类型	约束	描述
id	INTEGER	PRIMARY KEY AUTOINCREMENT	观看历史记录唯一标识符
user_id	INTEGER	FOREIGN KEY	关联到 users 表的 id
content_id	TEXT		视频内容 ID
watch_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	观看时间
video_title	TEXT		视频标题
video_desc	TEXT		视频描述
video_thumbnail	TEXT		视频缩略图 URL

关键代码 (CREATE_TABLE_WATCH_HISTORY):

Java

```java
private static final String CREATE_TABLE_WATCH_HISTORY = "CREATE TABLE " + TABLE_WATCH_HISTORY + "("
        + KEY_HISTORY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
        + KEY_USER_FOREIGN_ID + " INTEGER,"
        + KEY_CONTENT_ID + " TEXT,"
        + KEY_WATCH_DATE + " DATETIME DEFAULT CURRENT_TIMESTAMP," // 观看时间
        + KEY_VIDEO_TITLE + " TEXT,"
        + KEY_VIDEO_DESC + " TEXT,"
        + KEY_VIDEO_THUMBNAIL + " TEXT,"
        + "FOREIGN KEY(" + KEY_USER_FOREIGN_ID + ") REFERENCES " + TABLE_USERS + "(" + KEY_USER_ID + ")"
        + ")";
```

这段代码定义了创建 watch_history 表的 SQL 语句。其中：

FOREIGN KEY(user_id) REFERENCES users(id) 表示 user_id 列是外键，引用了 users 表的 id 列，建立了表之间的关联。

2.3 favorites 表

列名	数据类型	约束	描述
id	INTEGER	PRIMARY KEY AUTOINCREMENT	收藏记录唯一标识符
user_id	INTEGER	FOREIGN KEY	关联到 users 表的 id
content_id	TEXT		视频内容 ID
favorite_date	DATETIME	DEFAULT CURRENT_TIMESTAMP	收藏时间
video_title	TEXT		视频标题
video_desc	TEXT		视频描述
video_thumbnail	TEXT		视频缩略图 URL

关键代码 (CREATE_TABLE_FAVORITES):

Java

```
private static final String CREATE_TABLE_FAVORITES = "CREATE TABLE " + TABLE_FAVORITES + "("
        + KEY_FAVORITE_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
        + KEY_USER_FOREIGN_ID + " INTEGER,"
        + KEY_CONTENT_ID + " TEXT,"
        + KEY_FAVORITE_DATE + " DATETIME DEFAULT CURRENT_TIMESTAMP," // 收藏时间
        + KEY_VIDEO_TITLE + " TEXT,"
        + KEY_VIDEO_DESC + " TEXT,"
        + KEY_VIDEO_THUMBNAIL + " TEXT,"
        + "FOREIGN KEY(" + KEY_USER_FOREIGN_ID + ") REFERENCES " + TABLE_USERS + "(" + KEY_USER_ID + ")"
        + ")";
```

3. 数据库操作辅助类

代码中定义了五个数据库操作辅助类，它们都继承自 MyDBHelper：

UserDBHelper: 处理用户数据的增删改查。

WatchHistoryDBHelper: 处理观看历史数据的添加和查询。

FavoriteDBHelper: 处理收藏数据的添加、查询和删除。

ScoreDBHelper: 处理积分数据的查询、增加和减少。

MyDBHelper: 提供了数据库创建和升级的通用方法。

3.1 关键方法解释

MyDBHelper.onCreate(SQLiteDatabase db): 当数据库首次创建时调用，执行创建表的 SQL 语句。

Java

```java
@Override
public void onCreate(SQLiteDatabase db) {
    // 创建表
    db.execSQL(CREATE_TABLE_USERS);
    db.execSQL(CREATE_TABLE_WATCH_HISTORY);
    db.execSQL(CREATE_TABLE_FAVORITES);
    db.execSQL(CREATE_TABLE_SCORE);
}
```

MyDBHelper.onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion): 当数据库版本升级时调用，通常用于删除旧表并创建新表。

Java

```java
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersio
n) {
    // 如果需要升级数据库，可以在这里处理
    // 通常是删除旧表，然后创建新表
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_USERS);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_WATCH_HISTORY);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_FAVORITES);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_SCORE);
    onCreate(db);
}
```

UserDBHelper.addUser(User user): 添加新用户，使用 ContentValues 存储用户信息，并通过 db.insert() 方法插入数据库。

Java

```java
public boolean addUser(User user) {
    // ...
    ContentValues values = new ContentValues();
    values.put(KEY_USERNAME, user.getUsername());
    // ...
    long id = db.insert(TABLE_USERS, null, values);
    // ...
}
```

WatchHistoryDBHelper.getWatchHistory(int userId): 根据用户 ID 查询观看历史，使用 db.query() 方法，并根据 watch_date 倒序排列。

```java
public List<WatchHistory> getWatchHistory(int userId) {
    // ...
    Cursor cursor = db.query(TABLE_WATCH_HISTORY, new String[] {
            // ...
    }, KEY_USER_FOREIGN_ID + "=?", new String[] { String.valueOf(userId) }, null, null, KEY_WATCH_DATE + " DESC", null);
    // ...
}
```

FavoriteDBHelper.deleteFavorite(int userId, String contentId): 根据用户 ID 和内容 ID 删除收藏记录。

```java
public void deleteFavorite(int userId, String contentId) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_FAVORITES, KEY_USER_FOREIGN_ID + " = ? AND " + KEY_CONTENT_ID + " = ?", new String[]{String.valueOf(userId), contentId});
    db.close();
}
```

这份数据库设计方案通过四个表清晰地组织了用户信息、观看历史、收藏和积分数据。外键的运用确保了数据的一致性和完整性。五个辅助类提供了便捷的数据库操作方法，使应用的开发更加高效。该设计方案结构合理，易于理解和维护，能够满足应用的基本数据存储需求。

登录注册组件

课程模块

1. 获取数据

创建一个 ListView 用来展示视频列表

创建一个 CourseBean 用来存储各个 item 的详情信息

使用 HttpURLConnection 来打开事先编写好的 json 文件

```java
HttpURLConnection connection = (HttpURLConnection) new URL(url).openConnection();
        connection.setRequestMethod("GET");
        connection.connect();
```

注意：1. 由于使用的是 http 而非 https 所一在运行是会提醒存在安全问题，解决方法是在 xml 文件中加入编写：

```java
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="true" />
</network-security-config>
```

的文件即可。

在获取到 json 数据之后，解析 json 数据，并通过 listdata 保存至 CourseBean 中，最后返回 listdata

2. 访问需要创建线程，在子线程中进行

2.ListViewAdapter

缓存列表项中的各个子控件引用

```java
    static class ViewHolder {
        TextView tv_title, tv_loading;
        ImageView iv_img;
    }
```

1. 初始化 ViewHolder，绑定视图中的各个 UI，将 CourseBean 中的各项数据分别写入各项 UI 中

2. 通过传输的 ViewId 来拼接 uri，并访问服务器，通过日志来表示是否访问成功，成功后将获取的图片放入 ImageView 中

Java

```java
Glide.with(context)
    .load(uriTest)
    .listener(new RequestListener<Drawable>() {
        @Override
        public boolean onLoadFailed(@Nullable GlideException e, Object model, Target<Drawable> target, boolean isFirstResource) {
            Log.e("Glide", "Load failed", e);
            holder.tv_loading.setVisibility(View.GONE);
            return false;
        }

        @Override
        public boolean onResourceReady(Drawable resource, Object model, Target<Drawable> target, DataSource dataSource, boolean isFirstResource) {
            Log.d("Glide", "Load successful");
            holder.tv_loading.setVisibility(View.GONE);
            return false;
        }
    })
    .into(holder.iv_img);
```

3. 适配 CourseFragment

1. 初始化绑定对象

Java

```java
    binding = FragmentCourseBinding.inflate(inflater, container, false);
    View root = binding.getRoot();
    ListView listView = binding.lvCourse;
```

2. 初始化 ViewModel，观察 LiveData 并在数据变化时更新 UI

Java

```
    CourseViewModel viewModel = new ViewModelProvider(this).get(CourseViewModel.class);

    viewModel.getCourses().observe(getViewLifecycleOwner(), courses -> {
        listData.clear();
        listData.addAll(courses);
        listAdapter = new ListViewAdapter(getContext(), listData);
        listView.setAdapter(listAdapter);
        setupListViewClickListener();
    });
```

3. 动态获取选中的 CourseBean 对象，在选取后构建 WatchHistory 对象

Java

```
    CourseBean selectedCourseBean = listData.get(position);
    @SuppressLint("DefaultLocale")
    String uriText = String.format("http://159.75.231.207:9000/red/video/v_%d.png", (position + 1));
    WatchHistoryDBHelper watchHistoryDBHelper = new WatchHistoryDBHelper(getContext());
    WatchHistory watchHistory = new WatchHistory(
            user.getId(),
            selectedCourseBean.getId(),
            selectedCourseBean.getTitle(),
            selectedCourseBean.getDesc(),
            uriText
    );
    watchHistoryDBHelper.addWatchHistory(watchHistory);
```

4. 创建 Intent 并设置要传递的数据，启动新 Activity

Java

```
    Intent intent = new Intent(getContext(), VideoPlaybackActivity.class);
    intent.putExtra("position", position + 1); // 传递位置
    intent.putExtra("id", String.valueOf(selectedCourseBean.getId())); // 传递 ID
    intent.putExtra("title", selectedCourseBean.getTitle()); // 传递标题
    intent.putExtra("desc", selectedCourseBean.getDesc()); // 传递描述

    startActivity(intent);
```

注意：在完成 ListView 后，在虚拟机上可能无法加载出封面，可以在手机上进行调试

4. 创建 VideoPlaybackActivity

1. 创建一个全屏的遮罩层，将遮罩层添加到根布局，用于过度，使用 Handler 实现 0.5 秒的延迟

Java

```
        View overlay = new View(this);
        overlay.setBackgroundColor(Color.WHITE);
        overlay.setLayoutParams(new FrameLayout.LayoutParams(
                FrameLayout.LayoutParams.MATCH_PARENT,
                FrameLayout.LayoutParams.MATCH_PARENT
        ));
        new Handler().postDelayed(() -> {
            // 延迟结束，移除遮罩层
            ((RelativeLayout) findViewById(R.id.root_layout)).removeView(overlay);
        }, 500); // 500 毫秒即 0.5 秒
    }
```

2.initViewsAndData 的编写：初始化缓存

Java

```
        SharedPreferencesLoadUser sharedPreferencesLoadUser = new SharedPreferencesLoadUser(getSharedPreferences("data", MODE_PRIVATE));
        user = sharedPreferencesLoadUser.getUser();

        // 初始化缓存（如果尚未初始化）
        if (simpleCache == null) {
            File downloadDirectory = new File(getExternalCacheDir(), "media");
            try {
                simpleCache = new SimpleCache(downloadDirectory, new LeastRecentlyUsedCacheEvictor(1024 * 1024 * 100)); // 100MB cache
            } catch (Exception e) {
                Log.e(TAG, "Error initializing cache", e);
            }
        }
```

3. 获取从 CourseFragment 传递过来的位置和其他信息，并设置视频标题和描述

Java

```
    Intent intent = getIntent();
       id = Integer.parseInt(intent.getStringExtra("id"));
       strId = String.valueOf(id);
       desc = intent.getStringExtra("desc");
       title = intent.getStringExtra("title");

       TextView tvTitle = findViewById(R.id.tv_video_title);
       TextView tvDesc = findViewById(R.id.tv_video_desc);
       ImageView ivVideoStar = findViewById(R.id.iv_video_star);

       tvTitle.setText(title);
       tvDesc.setText(desc);
```

4.SharedPreferences 加载收藏状态，从 SharedPreferences 读取最新的收藏状态，编写收藏代码用 isFavorite 来追踪，保存收藏状态

Java

```java
        boolean isFavorite = loadFavoriteStatus(id);
        ImageView ivVideoStar = findViewById(R.id.iv_video_star);

        if (!isFavorite) {
            ivVideoStar.setImageResource(R.drawable.baseline_star_rate_24); // 已收藏图标
            favoriteDBHelper.addFavorite(watchHistory);
            saveFavoriteStatus(id, true);
            Toast.makeText(this, "已收藏", Toast.LENGTH_SHORT).show();
        } else {
            ivVideoStar.setImageResource(R.drawable.baseline_star_outline_24); // 未收藏图标
            favoriteDBHelper.deleteFavorite(user.getId(), strId);
            saveFavoriteStatus(id, false);
            Toast.makeText(this, "取消收藏", Toast.LENGTH_SHORT).show();
        }
    /**
     * 保存收藏状态
     */
    private void saveFavoriteStatus(int position, boolean isFavorite) {
        SharedPreferences prefs = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);
        SharedPreferences.Editor editor = prefs.edit();
        String key = generatePreferenceKey(position);
        editor.putBoolean(key, isFavorite);
        editor.apply();
    }
    /**
     * 加载收藏状态
     */
    private boolean loadFavoriteStatus(int position) {
        SharedPreferences prefs = getSharedPreferences(PREFS_NAME, MODE_PRIVATE);
        String key = generatePreferenceKey(position);
        return prefs.getBoolean(key, false);
    }
```

5. 初始化视频播放器（使用的是 exoplay，而非 ViewPage，exoplay 的基础功能相比于 ViewPager 更多，但不支持全局播放，和返回功能）

Java

```java
private void initializePlayer() {
        try {
            if (player == null) {
                player = new ExoPlayer.Builder(this).build();
                playerView.setPlayer(player);
                uriTest = String.format("http://159.75.231.207:9000/red/video/v_%d.mp4", id);

                String videoUrl = String.format(uriTest);
                Uri videoUri = Uri.parse(videoUrl);

                DataSource.Factory upstreamFactory = new DefaultHttpDataSource.Factory();
                CacheDataSource.Factory cacheDataSourceFactory =
                        new CacheDataSource.Factory()
                                .setCache(simpleCache)
                                .setUpstreamDataSourceFactory(upstreamFactory);

                ProgressiveMediaSource mediaSource = new ProgressiveMediaSource.Factory(cacheDataSourceFactory)
                        .createMediaSource(MediaItem.fromUri(videoUri));

                player.setMediaSource(mediaSource);
                player.prepare();
                player.setPlayWhenReady(true);
            }
        } catch (Exception e) {
            Log.e(TAG, "Error initializing player", e);
        }
    }
```

6. 监听设备状态，切换横竖屏，横屏时隐藏任务栏优化界面

Java

```
        adjustUIForOrientation(getResources().getConfiguration().orientation);
        hideStatusBarOnLandscape(getResources().getConfiguration().orientation);
    private void adjustUIForOrientation(int orientation) {
        if (orientation == Configuration.ORIENTATION_LANDSCAPE) {
            detailsLayout.setVisibility(View.GONE);
        } else {
            detailsLayout.setVisibility(View.VISIBLE);
        }
    }
    @Override
    public void onConfigurationChanged(@NonNull Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
        adjustUIForOrientation(newConfig.orientation);
        hideStatusBarOnLandscape(newConfig.orientation);
    }
```

习题模块

个人模块

1. 总体设计

个人信息用户界面主要包含以下几个部分：

用户信息展示：显示用户的头像、用户名、用户 ID。

功能入口：提供进入设置、历史记录和收藏页面的入口。

设置页面：包含修改个人信息、修改密码、账号与安全、隐私、关于、退出登录、注销账户等功能。

2. 页面结构与功能

2.1 MeFragment (主界面)

布局文件：fragment_me.xml (通过 FragmentMeBinding 绑定)

功能：

显示用户基本信息 (用户名、用户 ID)。

提供 “设置”、“历史记录” 和 “收藏” 的入口。

关键代码解释：

onAttach() 方法：

```java
@Override
public void onAttach(@NonNull Context context) {
    super.onAttach(context);
    // 初始化 SharedPreferences 和 UserDBHelper
    sharedPreferences = context.getSharedPreferences("data", Context.MODE_PRIVATE);
    userId = sharedPreferences.getInt("user_id", -1);
    userDBHelper = new UserDBHelper(context);
}
```

- 这段代码在 Fragment 与 Activity 关联时执行，用于初始化 SharedPreferences 和 UserDBHelper。

从 SharedPreferences 中获取用户 ID (user_id)，并使用 UserDBHelper 与数据库交互。

使用 onAttach() 方法可以确保在 onCreateView() 之前完成数据库帮助类的初始化。

onCreateView() 方法：

Java

```java
public View onCreateView(@NonNull LayoutInflater inflater,
                         ViewGroup container, Bundle savedInstanceStat
e) {
    // ...
    // 检查用户是否已登录
    if (userId != -1) {
        user = userDBHelper.getUserById(userId);
        // 检查 user 是否为 null
        if (user != null) {
            TextView tvUsername = binding.tvUserName;
            tvUsername.setText(user.getUsername());
            TextView tvUserID = binding.tvUserId;
            tvUserID.setText("UID: " + user.getId());

        } else {
            // ...
        }
    }
    // ...
    ivSetting.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(getActivity(), SettingActivity.
class);
            startActivity(intent);
        }
    });
    // ...
}
```

- 这段代码在创建 Fragment 视图时执行。

根据 userId 是否为 -1 判断用户是否登录。

如果用户已登录，则从数据库中获取用户信息并显示在界面上。

设置 ImageView (设置图标) 的点击事件，点击后跳转到 SettingActivity。

设置 "历史记录" 和 "收藏" 的点击事件，分别跳转到 HistoryActivity 和 FavoriteActivity。

2.2 SettingActivity (设置页面)

布局文件：activity_setting.xml

功能：

提供修改个人信息、修改密码、账号与安全、隐私、关于、退出登录、注销账户等功能的入口。

关键代码解释：

onCreate() 方法：

Java

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    Toolbar toolbar = findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    // ...
    RelativeLayout changeUserInfo = findViewById(R.id.rl_change_user_info);
    RelativeLayout changePsw = findViewById(R.id.rl_change_psw);
    // ...
    Button logoutButton = findViewById(R.id.bt_logout);

    changeUserInfo.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // 处理账号与安全点击事件
            Intent intent = new Intent(SettingActivity.this, ChangeUserInfoActivity.class);
            startActivity(intent);
        }
    });
    changePsw.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // 处理账号与安全点击事件
            editPsw("修改密码");
        }
    });
    // ...
    logoutButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // 处理账号与安全点击事件
            AlertDialog.Builder builder = new AlertDialog.Builder(SettingActivity.this);
            builder.setTitle("退出登录")
                    .setMessage("确定要退出登录吗？")
                    .setPositiveButton("确定", (dialog, which) -> {
```

```
                        sharedPreferencesLoadUser.clearUser();
                        Toast.makeText(SettingActivity.this, "退出成
功", Toast.LENGTH_SHORT).show();
                        Intent intent = new Intent(SettingActivity.thi
s, LoginActivity.class);
                        // 清除任务栈并创建新任务
                        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
| Intent.FLAG_ACTIVITY_CLEAR_TASK);
                        startActivity(intent);
                    })
                    .setNegativeButton("取消", null)
                    .show();
        }
    });
}
```

- 设置 Toolbar，并启用返回按钮。

为各个功能入口设置点击事件监听器。

changeUserInfo 点击后跳转到 ChangeUserInfoActivity。

changePsw 点击后调用 editPsw() 方法弹出修改密码对话框。

logoutButton 点击后弹出退出登录确认对话框，确认后清除 SharedPreferences 中的用户信息，并跳转到 LoginActivity。

editPsw(String title) 方法：

Java

```java
private void editPsw(String title) {
    // ...
    final EditText etOldPassword = dialogView.findViewById(R.id.et_old_password);
    final EditText etNewPassword = dialogView.findViewById(R.id.et_new_password);
    final EditText etConfirmPassword = dialogView.findViewById(R.id.et_confirm_password);

    builder.setView(dialogView)
            .setTitle(title)
            .setPositiveButton("确定", (dialog, which) -> {
                String oldPassword = etOldPassword.getText().toString();
                String newPassword = etNewPassword.getText().toString();
                String confirmPassword = etConfirmPassword.getText().toString();
                // ... 密码校验逻辑
                if (!MD5Utils.md5(oldPassword).equals(user.getPassword())) {
                    Toast.makeText(this, "原密码错误", Toast.LENGTH_SHORT).show();
                } else {
                    user.setPassword(MD5Utils.md5(newPassword));
                    userDBHelper.updateUser(user);
                    Toast.makeText(this, "修改成功", Toast.LENGTH_SHORT).show();
                }
            });
    // ...
}
```

- 弹出修改密码对话框。

使用 MD5Utils.md5() 对密码进行 MD5 加密。

校验原密码，更新密码并保存到数据库。

2.3 ChangeUserInfoActivity (修改用户信息页面)

布局文件：activity_change_user_info.xml

功能：

显示用户的详细信息 (用户名、用户 ID、手机号、注册时间)。

允许用户修改用户名和手机号。

关键代码解释：

loadUserInfo() 方法：

Java

```java
public void loadUserInfo() {
    // ...
    user = userDBHelper.getUserById(sp.getInt("user_id", -1));

    TextView tvUsername = findViewById(R.id.tv_username);
    // ...

    if (user != null) {
        tvUsername.setText(user.getUsername());
        // ...
    }
    // ...
}
```

- 从 SharedPreferences 中获取 user_id，并从数据库中加载用户信息。

将用户信息显示在对应的 TextView 中。

showEditDialog(String title, String currentValue) 方法：

Java

```java
private void showEditDialog(String title, String currentValue) {
    // ...
    final EditText etInput = dialogView.findViewById(R.id.et_input);
    etInput.setText(currentValue);
    if (title.equals("用户名")) {
        builder.setView(dialogView)
                .setTitle("修改" + title)
                .setPositiveButton("确定", (dialog, which) -> {
                    String newValue = etInput.getText().toString();
                    user.setUsername(newValue);
                    userDBHelper.updateUser(user);
                    Toast.makeText(this, "修改成功", Toast.LENGTH_SHOR
T).show();
                })
                .setNegativeButton("取消", null);
    } else if (title.equals("手机号")) {
       // ...
    }
    // ...
}
```

- 弹出修改信息对话框 (用户名或手机号)。

根据 title 参数判断是修改用户名还是手机号。

获取用户输入的新值，更新 user 对象，并调用 userDBHelper.updateUser() 方法更新数据库。

2.4 HistoryActivity (历史记录页面)

布局文件 : activity_video_list.xml

功能 :

以列表形式展示用户的观看历史。

每条记录包含视频缩略图、标题和观看时间。

点击列表项可以跳转到视频播放页面 (VideoPlaybackActivity)。

关键代码解释 :

populateVideoList() 方法 :

```java
private void populateVideoList() {
    watchHistoryDBHelper = new WatchHistoryDBHelper(this);
    List<WatchHistory> historyList = watchHistoryDBHelper.getWatchHistory(user.getId());
    videoItemList.addAll(historyList);

    videoAdapter.notifyDataSetChanged();
}
```

- 创建 WatchHistoryDBHelper 实例。

调用 getWatchHistory() 方法从数据库中获取当前用户的观看历史。

将观看历史数据添加到 videoItemList 中，并通知 VideoAdapter 更新数据。

VideoAdapter 类：

负责将观看历史数据绑定到 RecyclerView 的列表项中。

使用 Glide 库加载视频缩略图。

在 onBindViewHolder() 方法中设置列表项的点击事件，点击后跳转到 VideoPlaybackActivity 并传递视频相关信息。

2.5 FavoriteActivity (收藏页面)

布局文件： activity_video_list.xml (与 HistoryActivity 共用)

功能：

以列表形式展示用户的收藏记录。

每条记录包含视频缩略图、标题和收藏时间。

点击列表项可以跳转到视频播放页面 (VideoPlaybackActivity)。

关键代码解释：

populateVideoList() 方法：

```java
private void populateVideoList() {
    favoriteDBHelper = new FavoriteDBHelper(this);
    List<WatchHistory> favoriteList = favoriteDBHelper.getFavorite(use
r.getId());
    videoItemList.addAll(favoriteList);

    videoAdapter.notifyDataSetChanged();
}
```

- 创建 FavoriteDBHelper 实例。

调用 getFavorite() 方法从数据库中获取当前用户的收藏记录。

将收藏记录数据添加到 videoItemList 中，并通知 VideoAdapter 更新数据。

2.6 DeleteAccountActivity (删除账户页面)

布局文件： activity_delete_account.xml

功能：

提供注销账户的功能。

弹出确认对话框，防止误操作。

注销后删除数据库中的用户数据，清除 SharedPreferences 中的用户信息，并跳转到 LoginActivity。

关键代码解释：

showLogoutConfirmationDialog() 方法：

Java

```java
private void showLogoutConfirmationDialog() {
    new AlertDialog.Builder(this)
            .setTitle("注销账号")
            .setMessage("注销后不可恢复，确定要继续吗？")
            .setPositiveButton("确认", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    userDBHelper.deleteUser(sharedPreferencesLoadUser.getUser().getId());
                    sharedPreferencesLoadUser.clearUser();
                    Toast.makeText(DeleteAccountActivity.this, "注销成功", Toast.LENGTH_SHORT).show();
                    Intent intent = new Intent(DeleteAccountActivity.this, LoginActivity.class);
                    // 清除任务栈并创建新任务
                    intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                    startActivity(intent);
                    // 添加注销逻辑，这里可以清除用户数据或者返回登录界面
                }
            })
            .setNegativeButton("取消", null)
            .show();
}
```

- 弹出注销账号确认对话框。

确认后调用 userDBHelper.deleteUser() 删除数据库中的用户数据。

调用 sharedPreferencesLoadUser.clearUser() 清除 SharedPreferences 中的用户信息。

跳转到 LoginActivity，并清除任务栈。

程序构建和发布

代码上传到 Github： https://github.com/Rain-kl/RedCourseAPP

采用 Github Action 自动构建和发布，

Action 代码详见 : https://github.com/Rain-kl/RedCourseAPP/blob/master/.github/workflows/release.yml

这个 GitHub Actions 工作流用于在推送标签 (tag) 到仓库时，自动构建和发布 Android 应用的 APK 文件。工作流分为两个主要作业：prepare 和 build。

该 GitHub Actions 工作流实现了一个完整的 CI/CD 流程，包括：

1. 根据当前日期和时间生成版本号。
2. 检出代码，设置 Java 环境，并构建项目。
3. 运行测试，确保代码质量。
4. 构建发布版本的 APK 文件。
5. 使用密钥对 APK 文件进行签名。
6. 创建 GitHub Release，并将签名的 APK 文件上传到 Release 中。
7. 将构建的 APK 文件上传到 GitHub Artifacts，方便下载和存档。

1. 环境变量和触发器 (env 和 on)

env:

main_project_module: app：定义了一个环境变量 main_project_module，值为 app，表示主应用模块的名称。

on:

push:：指定触发工作流的事件为 push 操作。

tags:：指定触发条件为推送到仓库的标签。

v*.*.*：表示只有当推送的标签符合 v 开头，后面跟随三个由点号分隔的数字的格式（例如 v1.2.3）时，才会触发工作流。

2. prepare 作业

runs-on: ubuntu-latest: 指定该作业在最新的 Ubuntu 操作系统上运行。

outputs:: 定义该作业的输出，供其他作业使用。

- version: ${{ steps.set-ver.outputs.version }}：将 set-ver 步骤的输出 version 作为作业的输出。

steps::

- **id: set-ver:** 设置一个 ID 为 set-ver 的步骤。

- run: |：运行一段 shell 脚本。
  - echo "version=$(date -d "8 hour" -u +3.%y.%m%d%H)" >> $GITHUB_OUTPUT：
    - date -d "8 hour" -u +3.%y.%m%d%H：获取当前日期和时间，并将其格式化为 3.YY.MMDDHH 的形式，例如 3.23.102715 表示 2023 年 10 月 27 日 15 时。注意：这里加了 8 小时，可能是为了调整时区。
    - 将格式化后的日期和时间赋值给变量 version，并将其输出到 $GITHUB_OUTPUT，以便其他步骤或作业可以访问。

代码片段：

YAML

```yaml
prepare:
  runs-on: ubuntu-latest
  outputs:
    version: ${{ steps.set-ver.outputs.version }}
  steps:
    - id: set-ver
      run: |
        echo "version=$(date -d "8 hour" -u +3.%y.%m%d%H)" >> $GITHUB_OUTPUT
```

3. build 作业

needs: prepare: 指定该作业依赖于 prepare 作业，只有在 prepare 作业成功完成后才会执行。

runs-on: ubuntu-latest: 指定该作业在最新的 Ubuntu 操作系统上运行。

env::

- VERSION: ${{ needs.prepare.outputs.version }}：将 prepare 作业的输出 version 赋值给环境变量 VERSION。

steps::

- **uses: actions/checkout@v4:** 使用 actions/checkout@v4 检出代码仓库。
- **name: Set current date as env variable:** 设置当前日期为环境变量。
  - run: echo "date_today=$(date +'%Y-%m-%d')" >> $GITHUB_ENV：将当前日期以 YYYY-MM-DD 格式赋值给环境变量 date_today。

- **name: Set repository name as env variable:** 设置仓库名称为环境变量。
  - run: echo "repository_name=$(echo '${{ github.repository }}' | awk -F '/' '{print $2}')" >> $GITHUB_ENV：从 github.repository 变量中提取仓库名称，并赋值给环境变量 repository_name。
- **name: Set Up JDK:** 设置 JDK 环境。
  - uses: actions/setup-java@v4：使用 actions/setup-java@v4 设置 Java 环境。
  - with:：配置选项。
    - distribution: 'zulu'：指定 Java 发行版为 Zulu。
    - java-version: '17'：指定 Java 版本为 17。
    - cache: 'gradle'：启用 Gradle 缓存。
- **name: Change wrapper permissions:** 修改 Gradle wrapper 权限。
  - run: chmod +x ./gradlew：赋予 gradlew 文件可执行权限。
- **name: Run gradle tests:** 运行 Gradle 测试。
  - run: ./gradlew test：执行 Gradle test 任务。
- **name: Build gradle project:** 构建 Gradle 项目。
  - run: ./gradlew build：执行 Gradle build 任务。
- **name: Build apk release project (APK) - ${{ env.main_project_module }} module:** 构建发布版本的 APK 文件。
  - run: ./gradlew assembleRelease：执行 Gradle assembleRelease 任务，构建 app 模块的发布版本 APK。
- **uses: r0adkll/sign-android-release@v1:** 使用 r0adkll/sign-android-release@v1 对 APK 进行签名。
  - id: sign_app：设置步骤 ID 为 sign_app。
  - with:：配置选项。
    - releaseDirectory: ${{ env.main_project_module }}/build/outputs/apk/release：指定需要签名的 APK 文件所在的目录。
    - signingKeyBase64: ${{ secrets.SIGNING_KEY }}：指定签名密钥的 Base64 编码，存储在 GitHub Secrets 中。
    - alias: ${{ secrets.ALIAS }}：指定签名密钥的别名，存储在 GitHub Secrets 中。
    - keyStorePassword: ${{ secrets.KEY_STORE_PASSWORD }}：指定密钥库的密码，存储在 GitHub Secrets 中。
    - keyPassword: ${{ secrets.KEY_PASSWORD }}：指定签名密钥的密码，存储在 GitHub Secrets 中。

    - env:：环境变量。
        - BUILD_TOOLS_VERSION: "34.0.0"：指定构建工具的版本。
- **name: Create Release:** 创建 GitHub Release。
    - id: create_release：设置步骤 ID 为 create_release。
    - uses: actions/create-release@v1：使用 actions/create-release@v1 创建 Release。
    - env:：环境变量。
    - GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}：使用 GitHub 自动生成的 Token。
    - with:：配置选项。
        - tag_name: ${{ github.ref }}：指定 Release 的标签名称，使用触发工作流的标签名称。
        - release_name: Release ${{ github.ref }}：指定 Release 的名称。
        - draft: false：将 Release 设置为非草稿状态。
        - prerelease: false：将 Release 设置为非预发布版本。
- **name: Upload server release asset:** 上传已签名的 APK 文件到 GitHub Release。
    - uses: actions/upload-release-asset@v1：使用 actions/upload-release-asset@v1 上传 Release 资产。
    - env:：环境变量。
        - GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}：使用 GitHub 自动生成的 Token。
        - with:：配置选项。
        - upload_url: ${{ steps.create_release.outputs.upload_url }}：指定上传的 URL，使用 create_release 步骤的输出 upload_url。
        - asset_path: ${{ env.SIGNED_RELEASE_FILE }}：指定要上传的 APK 文件路径 ,${{ env.SIGNED_RELEASE_FILE }} 变量在应用签名后被设置，通常由 r0adkll/sign-android-release@v1 动作自动设置。
        - asset_name: ${{ env.repository_name }}-${{ env.VERSION }}.apk：指定上传后的文件名。
        - asset_content_type: application/vnd.android.package-archive：指定文件的 MIME 类型。