Часть 1. WebView

WebView в Android — это компонент, который позволяет разработчикам показывать веб-страницы внутри приложения. Это своего рода встроенный браузер, управляемый при помощи класса WebView. Он используется для отображения веб-контента, такого как пользовательские страницы для авторизации, справочные материалы или даже некоторые виды интерактивного контента внутри приложения.

Основные функции и использование WebView:

1. Отображение веб-страниц: Можно загружать HTML-страницы из интернета или загружать их как локальные ресурсы.
2. Интерактивность: Пользователи могут взаимодействовать со страницами, как в обычном веб-браузере.
3. JavaScript: WebView поддерживает выполнение JavaScript, что позволяет взаимодействовать с веб-страницей и обрабатывать пользовательский ввод.
4. Настройка и безопасность: Разработчики могут настраивать поведение WebView, включая управление кэшированием, cookies, историей просмотров и т.д. Важно правильно настроить параметры безопасности, чтобы избежать уязвимостей.

Рассмотрим пример использования WebView.

Сначала добавим WebView в XML макет Activity:

```
<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Затем нужно настроить WebView в коде Activity:

```
import android.os.Bundle;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import androidx.appcompat.app.AppCompatActivity;
```

```
    public class WebViewActivity extends AppCompatActivity {

        private WebView webView;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_web_view);

            webView = (WebView) findViewById(R.id.webview);
            webView.setWebViewClient(new  WebViewClient());  //
Помогает приложению открывать ссылки внутри WebView, а не во
внешнем браузере
            webView.getSettings().setJavaScriptEnabled(true); //
Включаем поддержку JavaScript

            webView.loadUrl("https://online-edu.mirea.ru "); //
Загрузка страницы
        }
    }
```

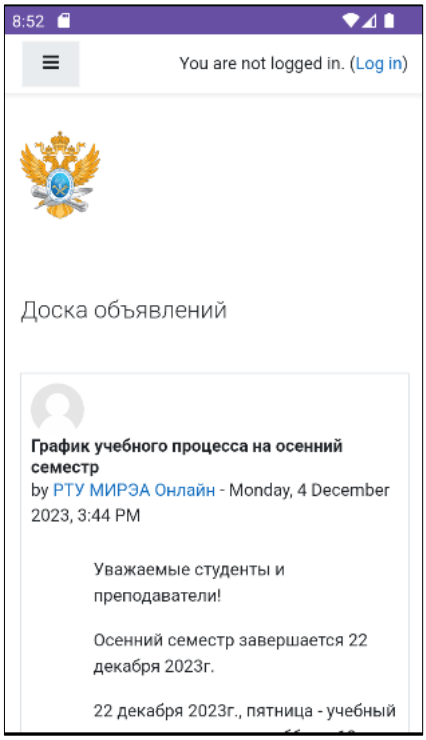
Важные моменты:

WebViewClient: Этот класс помогает управлять загрузкой различных URL и следить за процессом загрузки веб-страницы.

JavaScript: Включение JavaScript необходимо для многих современных веб-страниц, которые зависят от клиентских скриптов для пользовательской интерактивности.

Для получения доступа к интернету из приложения, необходимо указать в файле манифеста AndroidManifest.xml соответствующее **разрешение:**

```
<uses-permission android:name="android.permission.INTERNET"/>
```



Часть 2. Работа с мультимедиа

Работа с мультимедиа в Android — это важная часть разработки мобильных приложений, так как позволяет включать в приложения такие функции, как воспроизведение музыки, видео, захват изображений и видео с камеры, и обработка звука. Это улучшает интерактивность и функциональность приложения. Android предоставляет класс MediaPlayer для воспроизведения медиафайлов. Эти файлы могут быть локальными (хранятся на устройстве) или удаленными (доступны через интернет). Ранее был рассмотрен пример воспроизведения локального файла, теперь рассмотрим воспроизведение удаленного файла.

В код активности добавьте логику создания и управления MediaPlayer.

```
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    private MediaPlayer mediaPlayer;
```

```
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            Button playButton = findViewById(R.id.playButton);
            mediaPlayer = new MediaPlayer();
            try {
                mediaPlayer.setDataSource("URL вашего
аудиофайла");
                mediaPlayer.prepare(); // Можно использовать
prepareAsync() для сетевых потоков
            } catch (IOException e) {
                e.printStackTrace();
            }

            playButton.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    if (!mediaPlayer.isPlaying()) {
                        mediaPlayer.start();
                    } else {
                        mediaPlayer.pause();
                    }
                }
            });
        }

        @Override
        protected void onDestroy() {
            super.onDestroy();
            if (mediaPlayer != null) {
                mediaPlayer.release();
                mediaPlayer = null;
```

```
            }
        }
    }
```

Этот пример демонстрирует базовый способ воспроизведения аудио с использованием MediaPlayer. Обратите внимание на обработку исключений и корректное освобождение ресурсов в методе **onDestroy().**

Работая с мультимедиа, важно помнить о потенциальных исключениях и о необходимости управления ресурсами, чтобы избежать утечек памяти и сохранить стабильность приложения.

Часть 3. Анимации в Android

Анимации в Android используются для улучшения пользовательского интерфейса путём добавления визуальных эффектов при взаимодействии с элементами приложения. Они могут помочь сделать приложение более динамичным и интересным, а также интуитивно понятным, подчеркивая изменения состояний, переходы между активностями или фрагментами и реакции на действия пользователя.

Типы анимации в Android:

1. **View анимации:** Это простые анимации, которые можно применять к элементам интерфейса (views), такие как повороты, масштабирование, смещения и прозрачность. Они описываются в XML и могут быть запущены в коде.
2. **Анимация свойств (Property Animations):** Эти анимации предоставляют более сложный контроль, позволяя анимировать любое свойство объекта, такое как цвет фона, позиция или размер. Наиболее популярным классом для таких анимаций является ObjectAnimator.
3. **Анимации переходов:** Используются для анимации переходов между активностями или фрагментами, позволяя создавать сложные анимации при переключении между экранами.

Рассмотрим несколько примеров анимации элементов:

- Анимация вращения элемента.

```
import android.animation.ObjectAnimator;
import android.os.Bundle;
import android.widget.ImageView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ImageView imageView =
findViewById(R.id.rotateImageView);
        ObjectAnimator rotateAnim =
ObjectAnimator.ofFloat(imageView, "rotation", 0f, 360f);
        rotateAnim.setDuration(2000);
        rotateAnim.setRepeatCount(ObjectAnimator.INFINITE);
        rotateAnim.setRepeatMode(ObjectAnimator.RESTART);
        rotateAnim.start();
    }
}
```

ObjectAnimator – это подкласс **ValueAnimator**, который позволяет нам устанавливать целевой объект и свойство объекта для анимации. Это простой способ изменить свойства вида с указанной продолжительностью. Мы можем указать конечную позицию и продолжительность анимации.

Методы:

– **ofFloat()** – создает и возвращает ObjectAnimation, который анимирует координаты.

– **setDuration()** – устанавливает продолжительность анимации

– **getRepeatCount()** – определяет, сколько раз должна повторяться анимация.

– **getRepeatMode()** – определяет, что должна делать эта анимация, когда она дойдет до конца.

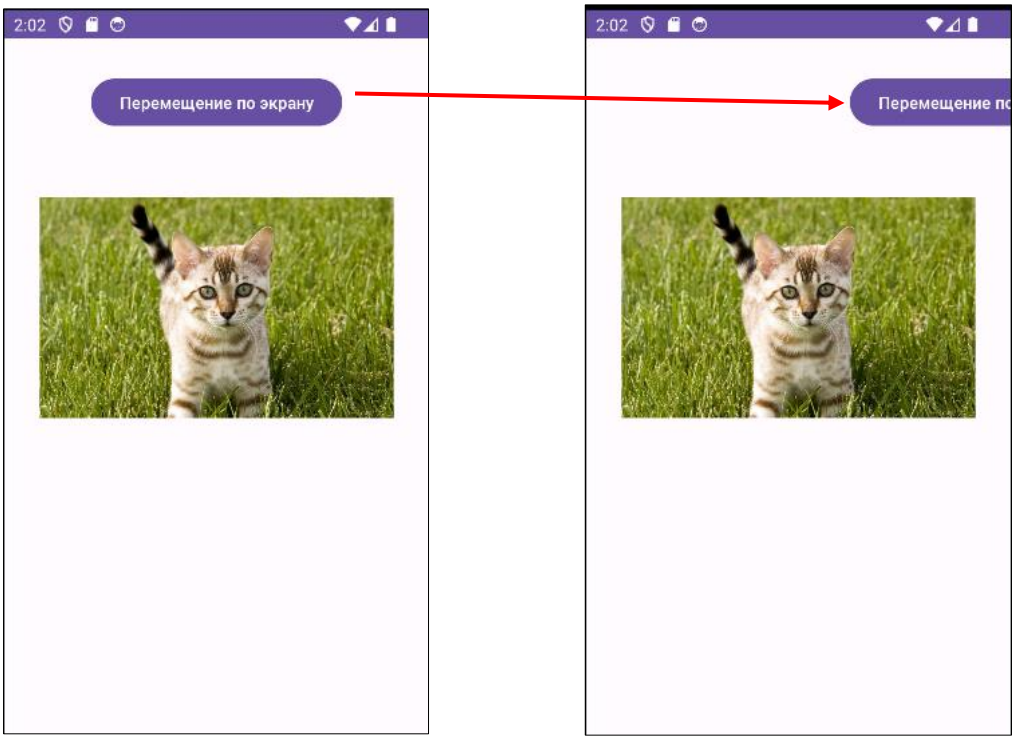
- Анимация перемещения элемента по экрану.

```
import android.animation.ObjectAnimator;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import androidx.appcompat.app.AppCompatActivity;


public class MainActivity extends AppCompatActivity {
```

```
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            final Button moveButton =
findViewById(R.id.moveButton);
            moveButton.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    ObjectAnimator moveAnim =
ObjectAnimator.ofFloat(moveButton, "translationX", 0f, 300f);
                    moveAnim.setDuration(1000);
                    moveAnim.start();
                }
            });
        }
    }
```



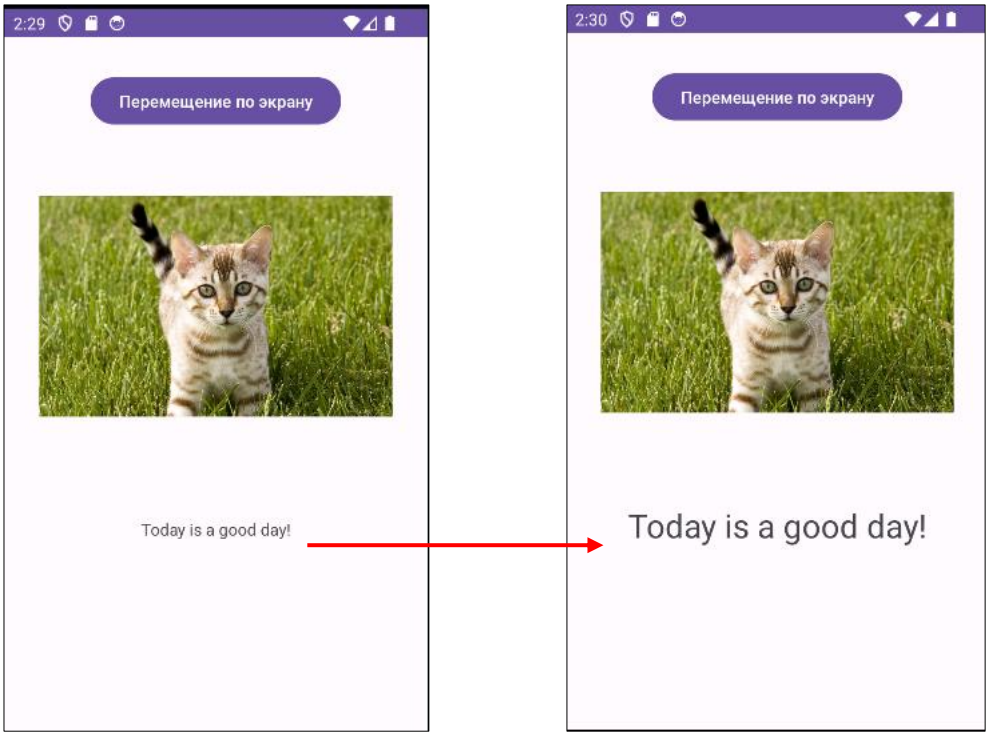
- Анимация изменения размера элемента.

```
import android.animation.ObjectAnimator;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView scaleText =
findViewById(R.id.scaleTextView);
        scaleText.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ObjectAnimator scaleX =
ObjectAnimator.ofFloat(scaleText, "scaleX", 1f, 2f);
                ObjectAnimator scaleY =
ObjectAnimator.ofFloat(scaleText, "scaleY", 1f, 2f);
                scaleX.setDuration(1000);
                scaleY.setDuration(1000);
                scaleX.start();
                scaleY.start();
            }
        });
    }
}
```



Эти примеры демонстрируют различные способы использования анимации для улучшения визуального восприятия и интерактивности Android-приложений.

Анимацию можно создавать как отдельный ресурс в папке под названием «anim» для хранения ресурсов анимации.

Часть 4. Уведомления

Уведомления в Android представляют собой сообщения, которые можно отображать вне интерфейса вашего приложения для информирования пользователей о различных событиях, даже когда приложение не используется активно. Это мощный инструмент для повышения взаимодействия и поддержания актуальности приложения в глазах пользователей.

Уведомления в Android управляются через NotificationManager, который является системной службой, ответственной за доставку уведомлений. Для создания уведомления используется Notification.Builder класс, который позволяет настроить заголовок, текст, иконку и другие параметры уведомления. С Android Oreo (API уровень 26) введено понятие каналов уведомлений (notification channels), которое требует определения категории

уведомлений, что позволяет пользователям управлять настройками уведомлений для различных типов контента.

Для отправки уведомлений на Android версии 13+ необходимо в файле манифеста прописать соответствующее разрешение.

```
<uses-permission
android:name="android.permission.POST_NOTIFICATIONS"/>
```

Далее создадим код для показа простых уведомлений.

```
    public class MainActivity extends AppCompatActivity {
        private static final String CHANNEL_ID =
"example_channel";

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            createNotificationChannel();

            final Button notifyButton =
findViewById(R.id.notifyButton);
            notifyButton.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    NotificationCompat.Builder builder = new
NotificationCompat.Builder(MainActivity.this, CHANNEL_ID)

.setSmallIcon(R.drawable.ic_notification)
                            .setContentTitle("Example
Notification")
                            .setContentText("This is a test
notification")

.setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

```
                    NotificationManager notificationManager =
getSystemService(NotificationManager.class);
                    notificationManager.notify(1,
builder.build());
                }
            });
        }

        private void createNotificationChannel() {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
{
                CharSequence name = "Example Channel";
                String description = "Channel for example
notifications";
                int importance =
NotificationManager.IMPORTANCE_DEFAULT;
                NotificationChannel channel = new
NotificationChannel(CHANNEL_ID, name, importance);
                channel.setDescription(description);
                NotificationManager notificationManager =
getSystemService(NotificationManager.class);

notificationManager.createNotificationChannel(channel);
            }
        }
    }
```

В этом примере создается канал уведомлений, что необходимо для API 26 и выше. Затем, при нажатии на кнопку, создается уведомление с использованием NotificationCompat.Builder, которое отображает текст и иконку. Уведомление отправляется через NotificationManager.

Используемая переменная CHANNEL_ID должна быть задана глобально в классе `private static final String CHANNEL_ID = "example_channel".`

Отложенные уведомления в Android позволяют отправить уведомление через определённый период времени, что может быть полезно для напоминаний, задач по расписанию или для других сценариев, где требуется не немедленное, а запланированное уведомление. Для реализации отложенных уведомлений можно использовать AlarmManager, который позволяет запланировать выполнение кода в определённое время.

Ниже приведен код активности для отправки отложенных уведомлений.

```
    public class MainActivity extends AppCompatActivity {
        private static final String CHANNEL_ID =
"delayed_channel";

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);

            createNotificationChannel();

            final Button delayedNotifyButton =
findViewById(R.id.delayedNotifyButton);
            delayedNotifyButton.setOnClickListener(new
View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    scheduleNotification(10000); // 10 секунд
                }
            });
        }

        private void createNotificationChannel() {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O)
{
                CharSequence name = "Delayed Notifications";
```

```
                String description = "Channel for delayed example
notifications";
                int                 importance                =
NotificationManager.IMPORTANCE_DEFAULT;
                NotificationChannel      channel      =      new
NotificationChannel(CHANNEL_ID, name, importance);
                channel.setDescription(description);
                NotificationManager    notificationManager    =
getSystemService(NotificationManager.class);

notificationManager.createNotificationChannel(channel);
            }
        }

        private void scheduleNotification(long delay) {
            Intent   notificationIntent   =   new   Intent(this,
AlarmReceiver.class);
            PendingIntent            pendingIntent            =
PendingIntent.getBroadcast(this,      0,      notificationIntent,
PendingIntent.FLAG_UPDATE_CURRENT                                |
PendingIntent.FLAG_IMMUTABLE);

            AlarmManager    alarmManager    =    (AlarmManager)
getSystemService(ALARM_SERVICE);
            long  futureInMillis  =  System.currentTimeMillis()  +
delay;
            alarmManager.setExact(AlarmManager.RTC_WAKEUP,
futureInMillis, pendingIntent);
        }
    }
```

Далее необходимо создать отдельный класс, который будет срабатывать при получении уведомления и создавать его.

```
    import android.content.BroadcastReceiver;
    import android.content.Context;
    import android.content.Intent;
```

```
import android.app.NotificationManager;
import androidx.core.app.NotificationCompat;

public class AlarmReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        NotificationCompat.Builder builder = new
NotificationCompat.Builder(context, MainActivity.CHANNEL_ID)
                .setSmallIcon(R.drawable.ic_notification)
                .setContentTitle("Delayed Notification")
                .setContentText("This is your scheduled
notification.")

.setPriority(NotificationCompat.PRIORITY_DEFAULT);

        NotificationManager notificationManager =
(NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(1, builder.build());
    }
}
```

Также его необходимо добавить в манифест.

```
<application ... >
    <receiver android:name=".AlarmReceiver" />
</application>
```

Таким образом, пользователь получит уведомление через 10 секунд после нажатия кнопки.

Задание

1. Реализовать в приложении просмотр любой веб-страницы через WebView.

2. Реализовать проигрывание музыки из интернета.

3. Реализовать различные анимации элементов UI.

4. Реализовать отправку уведомлений: обычную отправку и отложенную отправку

Источники

1) https://developer.android.com/develop/ui/views/layout/webapps/webview
2) https://quoality.medium.com/webview-1142072e6217
3) https://www.geeksforgeeks.org/how-to-play-audio-from-url-in-android/
4) https://metanit.com/java/android/11.2.php?ysclid=lv2d36rrw1266479582
5) https://developer.android.com/studio/write/motion-editor
6) https://medium.com/dsc-sastra-deemed-to-be-university/custom-animations-in-android-studio-4c96abfb5497
7) https://developer.android.com/develop/ui/views/notifications/build-notification
8) https://developer.alexanderklimov.ru/android/notification.php?ysclid=lv2d4hspyy756836933