👨‍💻 دليل المطور: ميزات التفاعل مع وسائل التواصل الاجتماعي

📚 فهرس المحتويات

🏗️ نظرة عامة على البنية

الهيكل العام للمشروع

```
app/src/main/java/com/animecharacter/
├── services/
│   ├── SocialMediaNotificationService.kt
│   ├── VisualEffectsService.kt
│   ├── AdvancedCharacterAnimationService.kt
│   └── AdvancedVoiceInteractionService.kt
├── managers/
│   ├── SocialMediaAchievementManager.kt
│   └── SocialMediaIntegrationManager.kt
├── models/
│   └── SocialMediaModels.kt
├── testing/
│   └── SocialMediaFeaturesTest.kt
└── utils/
    └── PreferencesHelper.kt (محدث)
```

تدفق البيانات

```
إشعار وسائل التواصل الاجتماعي
    ↓
SocialMediaNotificationService (تحليل)
    ↓
SocialMediaIntegrationManager (تنسيق)
    ↓
┌──────────────────┬──────────────────┬──────────────────┐
│  Visual Effects  │    Animations    │  Voice Messages  │
│     Service      │     Service      │     Service      │
└──────────────────┴──────────────────┴──────────────────┘
    ↓
SocialMediaAchievementManager (تتبع الإنجازات)
```

⚙️ إعداد البيئة التطويرية

المتطلبات الأساسية

```gradle
// في build.gradle (Module: app)
dependencies {
    // مكتبات جديدة للميزات
    implementation 'com.airbnb.android:lottie:5.2.0'
    implementation 'androidx.room:room-runtime:2.4.3'
    implementation 'androidx.room:room-ktx:2.4.3'
    kapt 'androidx.room:room-compiler:2.4.3'

    // Coroutines للمعالجة غير المتزامنة
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4'

    // ViewModel و LiveData
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2'
    implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.6.2'
}
```

الأذونات المطلوبة

```xml
<!-- في AndroidManifest.xml -->
<uses-permission
android:name="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE" />
<uses-permission android:name="android.permission.ACCESS_NOTIFICATION_POLICY"
/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

تسجيل الخدمات

```
<!-- خدمة مراقبة الإشعارات -->
<service
    android:name=".services.SocialMediaNotificationService"
    android:label="@string/social_media_notification_service_label"
    android:exported="false"
    android:permission="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE">
    <intent-filter>
        <action
android:name="android.service.notification.NotificationListenerService" />
    </intent-filter>
</service>
```

🔧 الخدمات الأساسية

1. SocialMediaNotificationService

الغرض: مراقبة وتحليل إشعارات وسائل التواصل الاجتماعي

```
class SocialMediaNotificationService : NotificationListenerService() {

    override fun onNotificationPosted(sbn: StatusBarNotification) {
        // تحليل الإشعار
        val platform = SUPPORTED_PACKAGES[sbn.packageName]
        if (platform != null) {
            analyzeNotification(sbn, platform)
        }
    }

    private fun analyzeNotification(sbn: StatusBarNotification, platform:
SocialMediaPlatform) {
        // استخراج النص والعنوان
        val notification = sbn.notification
        val title =
notification.extras.getCharSequence(Notification.EXTRA_TITLE)?.toString()
        val text =
notification.extras.getCharSequence(Notification.EXTRA_TEXT)?.toString()

        // تحديد نوع التفاعل
        val interaction = determineInteractionType("$`title `$text", platform)

        // إرسال النتيجة
        if (interaction != null) {
            handleSocialMediaInteraction(interaction)
        }
    }
}
```

إضافة منصة جديدة:

```
// في SUPPORTED_PACKAGES
private val SUPPORTED_PACKAGES = mapOf(
    "com.newplatform.android" to SocialMediaPlatform.NEW_PLATFORM,
    // ... المنصات الأخرى
)

// إضافة أنماط تحليل جديدة
private val NEW_PLATFORM_PATTERNS = listOf(
    Pattern.compile("(\\d+)\\s*new\\s+connections", Pattern.CASE_INSENSITIVE),
    // ... أنماط أخرى
)
```

2. VisualEffectsService

الغرض: إدارة المؤثرات البصرية والاحتفالات

```
class VisualEffectsService(private val context: Context) {

    fun triggerInteractionEffect(interaction: SocialMediaInteraction, level:
Int) {
        val effects = getEffectsForInteraction(interaction, level)
        effects.forEach { playVisualEffect(it) }
    }

    private fun playHeartEffect(parent: ViewGroup, effect: VisualEffect) {
        // إنشاء وتحريك القلوب
        repeat(effect.intensity * 5) { index ->
            val heartView = createHeartView()
            parent.addView(heartView)
            animateHeart(heartView, effect.duration)
        }
    }
}
```

إضافة مؤثر جديد:

```
// إنشاء مؤثر جديد
private fun createCustomEffect(level: Int): VisualEffect {
    return VisualEffect(
        type = "custom_effect",
        duration = 3000L,
        intensity = level,
        color = "#FF5722",
        position = VisualEffect.Position.AROUND_CHARACTER
    )
}

// تنفيذ المؤثر
private suspend fun playCustomEffect(parent: ViewGroup, effect: VisualEffect) {
    // منطق المؤثر المخصص
}
```

3. AdvancedVoiceInteractionService

الغرض: إدارة التفاعل الصوتي والرسائل المحفزة

```
class AdvancedVoiceInteractionService(private val context: Context) :
TextToSpeech.OnInitListener {

    fun speakCelebrationMessage(interaction: SocialMediaInteraction, level:
Int) {
        val messageKey = "${interaction.type.name.lowercase()}_level_${level}"
        val messages = celebrationMessages[messageKey] ?: return

        val selectedMessage = messages[Random.nextInt(messages.size)]
        val personalizedMessage = personalizeMessage(selectedMessage,
interaction)

        speak(personalizedMessage)
    }
}
```

إضافة رسائل جديدة:

```
// في loadCelebrationMessages()
celebrationMessages["new_platform_level_1"] = listOf(
    "رائع! تفاعل جديد على المنصة الجديدة!",
    "ممتاز! المحتوى الخاص بك يلقى استحساناً!",
    // ... رسائل أخرى
)
```

📊 نماذج البيانات

SocialMediaInteraction

```
data class SocialMediaInteraction(
    val type: Type,
    val count: Int,
    val platform: SocialMediaPlatform,
    val timestamp: Long,
    val content: String,
    val celebrationLevel: Int = 0
) {
    enum class Type {
        NEW_FOLLOWERS,
        NEW_LIKES,
        NEW_COMMENTS,
        NEW_SHARES,
        NEW_VIEWS,
        CELEBRITY_INTERACTION,
        TRENDING_MENTION,
        MILESTONE_REACHED
    }
}
```

SocialMediaGoal

```
data class SocialMediaGoal(
    val id: String,
    val title: String,
    val description: String,
    val platform: SocialMediaPlatform,
    val type: SocialMediaInteraction.Type,
    val targetValue: Int,
    val currentValue: Int = 0,
    val deadline: Long? = null,
    val isCompleted: Boolean = false,
    val rewardAnimation: String? = null,
    val rewardMessage: String? = null
)
```

إضافة نموذج بيانات جديد

```
data class CustomFeature(
    val id: String,
    val name: String,
    val isEnabled: Boolean = true,
    val settings: Map<String, Any> = emptyMap()
)

// إضافة إلى SocialMediaSettings
data class SocialMediaSettings(
    // ... الحقول الموجودة
    val customFeatures: List<CustomFeature> = emptyList()
)
```

🧪 التكامل والاختبار

تشغيل الاختبارات

```
// في النشاط الرئيسي أو نشاط الاختبار
class TestActivity : AppCompatActivity() {

    private lateinit var featuresTest: SocialMediaFeaturesTest

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        featuresTest = SocialMediaFeaturesTest(this)

        // تشغيل الاختبارات
        lifecycleScope.launch {
            val results = featuresTest.runAllTests()
            displayTestResults(results)
        }
    }

    private fun displayTestResults(results:
List<SocialMediaFeaturesTest.TestResult>) {
        val report = featuresTest.generateTestReport()
        Log.d("TestResults", report)

        // عرض النتائج في واجهة المستخدم
        showTestDialog(report)
    }
}
```

اختبار مكون محدد

```
// اختبار خدمة المؤثرات البصرية
private suspend fun testVisualEffectsOnly() {
    val visualEffectsService = VisualEffectsService(context)

    val testInteraction = SocialMediaInteraction(
        type = SocialMediaInteraction.Type.NEW_FOLLOWERS,
        count = 10,
        platform = SocialMediaPlatform.INSTAGRAM,
        timestamp = System.currentTimeMillis(),
        content = "Test interaction"
    )

    visualEffectsService.triggerInteractionEffect(testInteraction, 2)
}
```

محاكاة الإشعارات للاختبار

```
// إرسال إشعار تجريبي
private fun simulateNotification() {
    val intent = Intent("com.animecharacter.SOCIAL_MEDIA_INTERACTION")
    intent.putExtra("interaction_type", "NEW_FOLLOWERS")
    intent.putExtra("interaction_count", 5)
    intent.putExtra("platform", "INSTAGRAM")
    intent.putExtra("timestamp", System.currentTimeMillis())

    sendBroadcast(intent)
}
```

➕ إضافة ميزات جديدة

إضافة نوع تفاعل جديد

1. **تحديث التعداد:**

```
enum class Type {
    // ... الأنواع الموجودة
    NEW_CUSTOM_INTERACTION // نوع جديد
}
```

1. **إضافة أنماط التحليل:**

```
private val CUSTOM_PATTERNS = listOf(
    Pattern.compile("(\\d+)\\s*custom\\s+interactions",
Pattern.CASE_INSENSITIVE)
)
```

1. **تحديث منطق التحليل:**

```
private fun determineInteractionType(content: String, platform:
SocialMediaPlatform): SocialMediaInteraction? {
    // ... المنطق الموجود

    // فحص النوع الجديد
    for (pattern in CUSTOM_PATTERNS) {
        val matcher = pattern.matcher(content)
        if (matcher.find()) {
            return SocialMediaInteraction(
                type = SocialMediaInteraction.Type.NEW_CUSTOM_INTERACTION,
                count = extractNumber(matcher) ?: 1,
                platform = platform,
                timestamp = System.currentTimeMillis(),
                content = content
            )
        }
    }

    return null
}
```

إضافة رسوم متحركة جديدة

1. **إضافة ملف الرسوم المتحركة:**

```
app/src/main/assets/animations/lottie/custom_animation.json
```

1. **تسجيل الرسوم المتحركة:**

```
// في loadAvailableAnimations()
availableAnimations["custom_animation"] = SpecialAnimation(
    name = "custom_animation",
    filePath = "$LOTTIE_DIR/custom_animation.json",
    duration = 3000,
    triggerCondition = SpecialAnimation.TriggerCondition(
        interactionType = SocialMediaInteraction.Type.NEW_CUSTOM_INTERACTION,
        minimumCount = 1
    )
)
```

إضافة مؤثر بصري جديد

```
// في getEffectsForInteraction()
SocialMediaInteraction.Type.NEW_CUSTOM_INTERACTION -> {
    effects.add(createCustomVisualEffect(level))
}

private fun createCustomVisualEffect(level: Int): VisualEffect {
    return VisualEffect(
        type = "custom_visual",
        duration = 2500L,
        intensity = level,
        color = "#9C27B0",
        position = VisualEffect.Position.FULL_SCREEN
    )
}
```

🐛 استكشاف الأخطاء

مشاكل شائعة وحلولها

1. خدمة الإشعارات لا تعمل

المشكلة: `SocialMediaNotificationService` لا تستقبل إشعارات

الحلول:

```
// فحص إذن الوصول للإشعارات
private fun checkNotificationAccess(): Boolean {
    val enabledListeners = Settings.Secure.getString(
        contentResolver,
        "enabled_notification_listeners"
    )
    return enabledListeners?.contains(packageName) == true
}

// توجيه المستخدم لتفعيل الإذن
private fun requestNotificationAccess() {
    val intent = Intent(Settings.ACTION_NOTIFICATION_LISTENER_SETTINGS)
    startActivity(intent)
}
```

2. المؤثرات البصرية لا تظهر

المشكلة: `VisualEffectsService` لا يعرض المؤثرات

الحل:

```
// التأكد من تعيين العرض الأساسي
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val rootView = findViewById<ViewGroup>(android.R.id.content)
        visualEffectsService.setParentView(rootView)
    }
}
```

3. التفاعل الصوتي لا يعمل

المشكلة: `AdvancedVoiceInteractionService` صامت

الحل:

```
// فحص حالة TTS
override fun onInit(status: Int) {
    if (status == TextToSpeech.SUCCESS) {
        // تعيين اللغة
        val result = textToSpeech?.setLanguage(Locale("ar"))
        if (result == TextToSpeech.LANG_MISSING_DATA) {
            // تحميل بيانات اللغة العربية
            val installIntent = Intent()
            installIntent.action = TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA
            startActivity(installIntent)
        }
    }
}
```

تسجيل الأخطاء (Logging)

```
// استخدام تسجيل مفصل للتطوير
class SocialMediaNotificationService : NotificationListenerService() {

    companion object {
        private const val TAG = "SocialMediaNotificationService"
        private const val DEBUG = BuildConfig.DEBUG
    }

    private fun logDebug(message: String) {
        if (DEBUG) {
            Log.d(TAG, message)
        }
    }

    private fun logError(message: String, throwable: Throwable? = null) {
        Log.e(TAG, message, throwable)
    }
}
```

أدوات التطوير

```
// إضافة أدوات تطوير للاختبار السريع
class DeveloperTools {

    companion object {
        fun simulateFollowersIncrease(context: Context, count: Int) {
            val intent = Intent("com.animecharacter.SOCIAL_MEDIA_INTERACTION")
            intent.putExtra("interaction_type", "NEW_FOLLOWERS")
            intent.putExtra("interaction_count", count)
            intent.putExtra("platform", "INSTAGRAM")
            intent.putExtra("timestamp", System.currentTimeMillis())
            context.sendBroadcast(intent)
        }

        fun testAllAnimations(animationService:
AdvancedCharacterAnimationService) {
            val animations = animationService.getUnlockedAnimations()
            animations.forEach { animation ->
                animationService.playSpecificAnimation(animation.name)
            }
        }
    }
}
```

📋 قائمة المراجعة للتطوير

قبل إضافة ميزة جديدة

- [ ] تحديد المتطلبات بوضوح
- [ ] تصميم نموذج البيانات
- [ ] إنشاء اختبارات الوحدة
- [ ] تحديث الوثائق

قبل النشر

- [ ] تشغيل جميع الاختبارات
- [ ] فحص الأداء والذاكرة
- [ ] اختبار على أجهزة مختلفة
- [ ] مراجعة الأذونات المطلوبة
- [ ] تحديث دليل المستخدم

بعد النشر

- [ ] مراقبة التقارير والأخطاء
- [ ] جمع ملاحظات المستخدمين
- [ ] تحليل الاستخدام
- [ ] التخطيط للتحديثات القادمة

هذا الدليل يوفر أساساً قوياً لتطوير وصيانة ميزات التفاعل مع وسائل التواصل الاجتماعي. للمزيد من المساعدة، راجع الكود المصدري والتعليقات المفصلة.

🗣️ ميزة اسم التفعيل الصوتي المخصص

تسمح هذه الميزة للمستخدم بتحديد كلمة تفعيل مخصصة للشخصية، بحيث تستجيب الشخصية للأوامر الصوتية فقط عند سماع هذه الكلمة.

المكونات الرئيسية:

1. **`PreferencesHelper.kt`**: تم إضافة دوال لحفظ واسترجاع كلمة التفعيل وحالة تفعيل الميزة. `kotlin // إعدادات اسم التفعيل fun getWakeWord(): String fun setWakeWord(wakeWord: String) fun isWakeWordEnabled(): Boolean fun setWakeWordEnabled(enabled: Boolean)`
2. **`AdvancedVoiceInteractionService.kt`**: تم تعديل هذه الخدمة لتشمل:
    - تهيئة `SpeechRecognizer` للاستماع المستمر.
    - دوال `startListeningForWakeWord()` و `stopListeningForWakeWord()` للتحكم في عملية الاستماع.
    - منطق `handleRecognizedText()` لمقارنة النص المتعّرف عليه بكلمة التفعيل المحددة في الإعدادات.
    - إضافة إذن `RECORD_AUDIO` و `BIND_RECOGNITION_SERVICE` في `AndroidManifest.xml`.
3. **`preferences.xml` و `SettingsActivity.kt`**: تم إضافة عناصر واجهة المستخدم للسماح للمستخدم بتفعيل/تعطيل الميزة وتحديد كلمة التفعيل.
    - `SwitchPreferenceCompat` لـ `is_wake_word_enabled`.

- `EditTextPreference` لـ `wake_word`.

كيفية الاختبار:

- **اختبار الوحدة:** تم إنشاء `WakeWordFeatureTest.kt` لاختبار منطق تفعيل/تعطيل كلمة التفعيل وبدء/إيقاف الاستماع.
- **الاختبار اليدوي:**
  1. افتح إعدادات التطبيق وانتقل إلى قسم الصوت.
  2. قم بتفعيل "تفعيل كلمة التفعيل" وأدخل كلمة تفعيل (مثال: "مُزنة").
  3. ارجع إلى الشاشة الرئيسية للتطبيق.
  4. قل كلمة التفعيل التي حددتها. يجب أن تستجيب الشخصية برسالة تأكيد.
  5. جرب قول كلمات أخرى لا تتضمن كلمة التفعيل. يجب ألا تستجيب الشخصية.

🎮 ميزة إخفاء الأيقونة العائمة أثناء الألعاب

تسمح هذه الميزة بإخفاء الأيقونة العائمة للشخصية تلقائياً عند تشغيل الألعاب المحددة من قبل المستخدم، وإعادة إظهارها عند الخروج من اللعبة.

المكونات الرئيسية:

1. **`PreferencesHelper.kt`**: تم إضافة دوال لحفظ واسترجاع حالة تفعيل الميزة وقائمة حزم الألعاب. `kotlin // إعدادات إخفاء الأيقونة أثناء الألعاب fun isHideOnGameEnabled(): Boolean fun setHideOnGameEnabled(enabled: Boolean) fun getGamePackageNames(): Set<String> fun setGamePackageNames(packageNames: Set<String>)`
2. **`Live2DFloatingWindowService.kt`**: تم إضافة دوال `showWindowProgrammatically()` و `hideWindowProgrammatically()` للتحكم في رؤية النافذة العائمة من خدمات أخرى.
3. **`AppMonitorService.kt`**: خدمة جديدة مسؤولة عن:
   - مراقبة التطبيق النشط في المقدمة باستخدام `ActivityManager`.
   - التحقق مما إذا كان التطبيق النشط هو أحد الألعاب المحددة في الإعدادات.
   - إخفاء أو إظهار النافذة العائمة بناءً على نتيجة التحقق.

4. **`AndroidManifest.xml`**: تم إضافة إذن `PACKAGE_USAGE_STATS` (يتطلب إذن خاص من المستخدم) وتسجيل `AppMonitorService`.
5. **`preferences.xml`** و **`SettingsActivity.kt`**: تم إضافة عناصر واجهة المستخدم للسماح للمستخدم بتفعيل/تعطيل الميزة وتحديد قائمة الألعاب.
    - `SwitchPreferenceCompat` لـ `hide_on_game_enabled`.
    - `MultiSelectListPreference` لـ `game_package_names` لعرض قائمة بالتطبيقات المثبتة واختيار الألعاب منها.

كيفية الاختبار:

- **اختبار الوحدة:** تم إنشاء `GameDetectionFeatureTest.kt` لاختبار منطق اكتشاف الألعاب وإخفاء/إظهار النافذة.
- **الاختبار اليدوي:**
    1. افتح إعدادات التطبيق وانتقل إلى قسم "إخفاء الأيقونة أثناء الألعاب".
    2. قم بتفعيل "إخفاء الأيقونة تلقائياً" وحدد بعض الألعاب المثبتة على جهازك.
    3. تأكد من أن الشخصية العائمة مرئية.
    4. افتح إحدى الألعاب التي حددتها. يجب أن تختفي الأيقونة العائمة.
    5. اخرج من اللعبة أو انتقل إلى تطبيق آخر. يجب أن تظهر الأيقونة العائمة مرة أخرى.