به نام خدا

موضوع پروژه: iCamera

کارکرد پروژه: مدیریت دسترسی اپلیکیشن ها

توسعه دهنده: حسین کرمی

امکانات پروژه: اپلیکیشن اندرویدی iCamera

زبان‌های توسعه: Kotlin و Java

نحوه کارکرد اپلیکیشن

به کمک این اپلیکیشن می‌توانید از آن به عنوان نرم افزار هوشمند دستگاه آیفون تصویری استفاده کرد. این اپلیکیشن امکان اتصال به چند دوربین، نمایش تصویر، باز کردن درب و... را داراست.

برنامه‌نویسی اپلیکیشن

برای برنامه‌نویسی این اپلیکیشن از دو زبان کاتلین و جاوا استفاده شده است. کاتلین زبان رسمی توسعه اپلیکیشن‌های اندروید است و سازگاری بسیار زیادی با زبان جاوا و کلاس‌ها و وابستگی‌های نوشته شده به زبان جاوا دارد.

همچنین زبان کاتلین برای مدیریت وابستگی‌ها از Gradle استفاده می‌کند.

iCamera ؛ یک اپلیکیشن سیستمی

برای اینکه این اپلیکیشن به امکانات کامل دسترسی داشته باشد، باید به صورت سیستمی تعریف شود.

* برای اینکار ابتدا دستگاه را روت کنید.
* سپس اپلیکیشن Link2SD را از گوگل پلی نصب کنید.
* سپس اپلیکیشن iCamera را از طریق اپلیکیشن پیدا کنید.
* انگشت خود را بر روی اپلیکیشن نگه دارید و Convert to system app را انتخاب کنید.

توضیحات کد پروژه

VideoProject.libuvccamera:

این ماژول امکان تبدیل سیگنال دوربین به تصویر قابل دیدن بر روی SurfaceView را فراهم میکند. به کمک این ماژول می‌توانید به راحتی اطلاعات را از طریق USB گرفته و اطلاعات را بر روی یک View نمایش داد.

به کد زیر در فرگمنت CameraFragment توجه کنید:

**mCameraHelper** = UVCCameraHelper().*apply* **{** setDefaultFrameFormat(UVCCameraHelper.*FRAME\_FORMAT\_MJPEG*)  
 initUSBMonitor(*activity*, **mUVCCameraView**, **listener**)  
**}**

در اینجا ما ابتدا USBMonitor را مقداردهی اولیه می‌کنیم و سپس از این Helper یک مقدار بازگردانده می‌شود. از این هلپر می‌توانیم برای مدیریت دوربین استفاده کنیم. لازم به ذکر است که تمام مراحل شناسایی USB و دریافت مجوز‌ها نیز به کمک همین ماژول قابل دسترسی است.

ذکر این نکته حائز اهمیت است که اگر اپلیکیشن سیستمی باشد، مجوز بدون نیاز به نمایش پیغام تایید مجوز، تایید خواهد شد.

کلاس MyAppliation.kt

در این کلاس موارد مهمی صورت می‌گیرد که پرداختن به آن‌ها حائز اهمیت است.

ارسال و دریافت دیتا (ورودی و خروجی) به کمک این کلاس صورت می‌گیرد. فرض کنید دستوراتی را بخواهید از یک USB ارسال یا دریافت کنید. برای اینکه دریافت و ارسال دیتا به صورت همگانی، عمومی و در کل اپلیکیشن صورت گیرد، بهتر است از کلاس اپلیکیشن برای این منظور استفاده کنیم.

کلاس اپلیکیشن به کلاسی گفته می‌شود که در همه جای اپلیکیشن معتبر است و کانکستی را در اختیار کل اپلیکیشن قرار می‌دهد که در همه جا معتبر است.

بنابراین اگر ارسال و دریافت اطلاعات را به کمک این کلاس انجام دهیم، دیگر نیازی به باز تعریف کد‌های ارسال و دریافت دیتا برای تک تک اکتیویتی‌ها نخواهیم بود.

به کد زیر توجه کنید:

**val** manager = getSystemService(Context.*USB\_SERVICE*) **as** UsbManager  
**val** availableDrivers =  
 UsbSerialProber.getDefaultProber().findAllDrivers(manager)  
**if** (availableDrivers.isEmpty()) {  
 **return**}  
  
**val** drivers =  
 availableDrivers.*filter* **{  
 it**.*device*.*let* **{** device **->** device.*productId* == 60000 && device.*vendorId* == 4292 **}  
 }  
  
if** (drivers.isEmpty())  
 **return  
  
val** driver = drivers[0]  
**val** connection = manager.openDevice(driver.*device*)  
**if** (connection == **null**) {  
 **val** mPermissionIntent = PendingIntent.getBroadcast(**this**, 0, Intent(**"1000"**), 0)  
 manager.requestPermission(driver.*device*, mPermissionIntent)  
 **return**}  
**try** {  
 **port** = driver.*ports*[0]  
 **port**!!.open(connection)  
 **port**!!.setParameters(9600, 8, UsbSerialPort.*STOPBITS\_1*, UsbSerialPort.*PARITY\_NONE*)  
} **catch** (e: IOException) {  
 e.printStackTrace()  
}  
**try** {  
 **val** usbIoManager = SerialInputOutputManager(**port**, **this**)  
 usbIoManager.start()  
} **catch** (e: Exception) {  
 e.printStackTrace()  
}

این کد امکان دسترسی به USB‌ها را فراهم می‌کند. به کمک این کد می‌توانید دستگاه‌ها را یافته و پورت را برای ارسال و دریافت دیتا بر روی ریت 9600 آماده سازی کرد.

متد onNewData، متدی است که با دریافت اطلاعات جدید از USB، این متد صدا زده خواهد شد.

مورد مهمی که در این اپلیکیشن پیاده سازی شده است، امکان مدیریت تاچ به کمک دریافت نقاط است.

به کد زیر توجه کنید:

**val** result = StringBuilder()  
**for** (bb **in** data) {  
 result.append(String.*format*(**"%02X"**, bb))  
}  
  
**val** point = result.toString()  
  
**when** {  
 **pointReceived**.*startsWith*(**"FFFF"**) && point.**length** == 8  
 || **pointReceived**.*startsWith*(**"FFFF"**) && **pointReceived**.**length** == 6 && point.**length** == 6  
 || **pointReceived**.*startsWith*(**"FFFF"**) && **pointReceived**.**length** == 8 && point.**length** == 4  
 || **pointReceived**.*startsWith*(**"FFFF"**) && **pointReceived**.**length** == 10 && point.**length** == 2 -> {  
 **pointReceived** += point  
  
 Log.d(**"Point Received"**, **pointReceived**)  
  
 **if** (!**critical** && **pointReceived**.**length** == 12) {  
 performSimulation()  
 } **else if** (!**critical**) {  
 **pointReceived** = **""** }  
  
 }  
  
 point == **"FF"** && **pointReceived**.*isEmpty*() -> {  
 **pointReceived** = **"FF"** }  
  
 point == **"FF"** && **pointReceived** == **"FF"** -> {  
 **pointReceived** = **"FFFF"** }  
  
 **pointReceived**.*startsWith*(**"FFFF"**) && **pointReceived**.**length** < 12 ->  
 **pointReceived** = **pointReceived**.plus(point)  
  
 **pointReceived**.*startsWith*(**"FFFF"**) && **pointReceived**.**length** > 12 -> {  
 **pointReceived** = **""** }

در کد بالا اگر نقطه‌ای ارسال شود (که با FF شروع می‌شود)، این نقطه بر روی اسکرین به صورت دینامیک تپ خواهد شد. pointReceived متغیری است که مقدار نقطه تپ شده را دریافت میکند.

پس از اینکه نقطه یافت شد، به کمک متد performSimulation این نقطه را بر روی اسکرین تپ می‌کنیم.

**private fun** performSimulation() {  
 **critical** = **true** Log.d(**"Performing Point "**, **pointReceived**)  
 **try** {  
 **val** rPoint = **pointReceived**.*replace*(**"FFFF"**, **""**)  
 **val** x = Integer.parseInt(rPoint.*substring*(2, 4) + rPoint.*substring*(0, 2), 16)  
 **val** y = Integer.parseInt(rPoint.*substring*(6) + rPoint.*substring*(4, 6), 16)  
  
 simulate2(x.toFloat(), y.toFloat())  
 *//simulateClick(x.toFloat(), y.toFloat())* } **catch** (e: Exception) {  
 e.printStackTrace()  
 }  
  
 **pointReceived** = **""  
 critical** = **false**}

به متد performSimulation دقت کنید. در این متد ما ابتدا نقطه را به حالتی که قابل تاچ به صورت x و y باشد تبدیل می‌کنیم. برای اینکار ابتدا باید FFFF را از ابتدا آن پاک کنیم. سپس x و y را جدا کرده و به تابع simulate2 بدهید.

به simulate2 دقت کنید:

**private fun** simulate2(fakeX: Float, fakeY: Float) {  
 **val** x = ((*resources*.*displayMetrics*.**widthPixels** / 1024.0) \* fakeX).toFloat()  
 **val** y = ((*resources*.*displayMetrics*.**heightPixels** / 600.0) \* fakeY).toFloat()  
 **object** : Thread() {  
 **override fun** run() {  
 **val** m\_Instrumentation = Instrumentation()  
 m\_Instrumentation.sendPointerSync(  
 MotionEvent.obtain(  
 SystemClock.uptimeMillis(),  
 SystemClock.uptimeMillis(),  
 MotionEvent.*ACTION\_DOWN*, x, y, 0  
 )  
 )  
 m\_Instrumentation.sendPointerSync(  
 MotionEvent.obtain(  
 SystemClock.uptimeMillis(),  
 SystemClock.uptimeMillis(),  
 MotionEvent.*ACTION\_UP*, x, y, 0  
 )  
 )  
 }  
 }.*apply* **{** start() **}**}

در این متد ما دو نقطه x و y را دریافت میکنیم. دقت فرمایید که اطلاعاتی که ما از USB می‌گیریم، برای صفحه 1024 در 600 پیکسل تنظیم شده است. برای اینکه با اندازه واقعی اسکرین این مقدارها را تبدیل کنیم، باید به نسبت صحیح تبدیل کرده و سپس عمل تپ کردن را انجام داد.

از displayMetrics می‌توانید سایز اسکرین را دریافت کنید. سپس مقدار x و y را به این نسبت تبدیل کرد. سپس می‌توان به کمک کلاس Instrumentation عمل تپ کردن را به شکلی که در کد می‌بینید، انجام دهید.

کلاس HotKeys:

این کلاس مقادیر دیتای معتبر ارسال و دریافت دیتا را در خود دارد.

**object** HotKeys {  
 **const val RECEIVE\_OPEN\_FIRST\_CAMERA** = **"b2"  
 const val SEND\_OPEN\_FIRST\_DOOR** = **"b3"  
 const val SEND\_EXIT** = **"b4"  
 const val SEND\_FIRST\_CAMERA\_SELECTED** = **"b5"  
 const val SEND\_SECOND\_CAMERA\_SELECTED** = **"b6"  
 const val SEND\_OPEN\_SECOND\_DOOR** = **"b7"  
 const val RECEIVE\_OPEN\_SECOND\_CAMERA** = **"b8"**}

کد بالا همان HotKeyها می‌باشد.

Android Manifest:

نکته مهمی که در مانیفست اندروید وجود دارد، لانچر شدن اپلیکیشن iCamera و تبدیل آن به نرم افزار لانچر دیفالت سیستم عامل است.

برای اینکار اگر به MainActivity دقت کنید، کد زیر را خواهید دید:

<**action android:name="android.intent.action.MAIN"** />  
<**category android:name="android.intent.category.HOME"** />  
<**category android:name="android.intent.category.DEFAULT"** />  
<**category android:name="android.intent.category.LAUNCHER"** />  
<**action android:name="android.intent.action.VIEW"** />

در کد بالا اپلیکیشن به صورت لانچر تعریف شده است. لانچر شدن اپلیکیشن این قابلیت را فراهم می‌کند که اپلیکیشن به صورت دیفالت پس از بوت شدن سیستم عامل اجرا شود.