**ارسال درخواست:**

برای فهم ایده­ای که چه کاری میخوایم انجام بدیم Placeholder فعلی مناسبه، ولی آلان زمان مناسبیه که دیتای واقعی رو درخواست کنیم و بعدا باهاش RecyclerViewمون رو اشغال کنیم. برای دستیابی به دیتا، ما از API OpenWeatherMap و چنتا کلاس معمولی استفاده میکنیم. همینطور که گفتیم، همکاری کاتلین با جاوا واقعا قویه، پس شما میتونین از هر کتابخونه که دوست دارین استفاده کنین، به عنوان مثال برای درخواست از سرور از کتابخونه Retrofit استفاده کنیم. هرچند چون ما داریم درخواست های ساده API رو انجام میدیم نیازی نیست که لایبری سوم شخصی رو به پروژمون اضافه کنیم.

علاوه بر این همینطور که بعدا میبینیم، کاتلین توابع الحاقی برامون فراهم کرده که ما میتونیم درخواست هامون رو خیلی راحت ارسال کنیم. بیاین یک کلاس ساده Request درست کنیم:

class Request(val url: String) {

fun run() {  
 val forecastJsonStr = URL(url).readText()  
 Log.d(javaClass.simpleName, forecastJsonStr)  
 }  
}

یک URL رو به صورت ورودی دریافت میکنه و نتیجه JSON رو توی Logcat چاپ میکنه. پیاده سازی این تابع با استفاده از تابع الحاقی readText() خیلی ساده شده، که این تابع هم توی Kotlin Standard Library موجوده. البته برای جواب­های بزرگ پیشنهاد نمیشه که از این تابع استفاده بشه ولی برای مورد ما به اندازه کافی خوبه.

اگه این تابع رو با اونی که توی جاوا نیاز داشتیم مقایسه کنین متوجه میشین که خیلی از بارپردازشی کم شده اونم تنها با استفاده از لایبری استاندارد کاتلین. اگه میخواستیم توی جاوا همین عمل رو انجام بدیم وجود یک HttpURLConnection و یک BufferedReader و یک حلقه برای پیدا کردن نتیجه الزامی بود تازه این­ها همه جدا از مدیریت status های ارتباطات و غیره و غیره است! مسلما این همین کاریه که این تابع پشت پرده انجام میده و اینو ما به راحتی در اختیار داریم.

برای این که برنامه بتونه درخواست رو ارسال کنه نیاز به اجازه دسترسی به اینترنت داریم، پس وارد AndroidManifest.xml میشیم و خط زیر رو بهش اضافه میکنیم:

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

**ارسال درخواست خارج از نخ اصلی :**

همینطور که میدونین درخواست HTTP اجازه نداره توی حلقه اصلی اجرا بشه،چرا که یک اکسپشن رو پرتاب میکنه! و همچنین میدونین که قفل کردن صفحه UI یک کار نادرستیه. راه حل معمول استفاده از AsyncTask هه ولی اگه به درستی از استفاده نشه میتونه خیلی خطرناک باشه! چون به مرور زمان به تابع postExecute میرسه و اون ممکنه Activity رو از بین ببره که باعث crash کردن اپ میشه.

Anko یک DSL خیلی ساده برای کار با asynchrony ها تهیه کرده که میتونه نیاز های ابتدایمون رو مهیا کنه. درواقع یک تابع async درست کرده که میتونه کدمون رو توی نخ دیگه انجام بده! و حتی این گزینه رو برامون فراهم کرده که هرموقع خواستیم به رشته اصلی برگردیم اونم با تنها استفاده از uiThread . اجرای کدمون توی نخ دوم به راحتی کد زیره:

async() **{** Request(url).run()  
 uiThread **{** longToast("Request performed") **}  
}**

یکی از ویژگی های جالب uiThread اینه که بسته به شی ای که صداش میزنه، مقداردهی میشه! اگه توی Activity صدا زده بشه uiThread اجرا نمیشه اگر activity.isFinishing() مقدار true برگردونه و درنتیجه باعث crash کردن برنامه شما نمیشه.

شما حتی میتونین اجرا کننده خودتون رو بنویسین :

val executor = Executors.newScheduledThreadPool(4)  
async(executor) **{** // Some task  
**}**

اگه شما بخواین با futures کار کنین، async یک Future جاوا برمیگردونه. حتی شما اگه بخواین یک Future با result رو برگردونین میتونین از asyncResult استفاده کنین!

واقعا سادس نه؟ و واقعا خواناتر از AsyncTasks هه. فعلا من یک URL ثابت رو به عنوان درخواست میفرستم تا مطمئن شم که نتیجه­ای که دریافت میکنم درسته و ایا من قادر هستم اینو توی Activity نشون بدم. راجع به نحوه pars کردن فایل های JSON و تبدیلشون به کلاس­های دیتا صحبت میکنیم ولی قبل از اون لازمه که بدونیم که اصلا کلاس­های دیتا چی هستن.