ملخص أدوات مهام الخلفية

مفهوم Service

متطلبات التشغيل	جميع نسخ الأندرويد
رابط المكتبة على Gradle	موجود بشكل افتراضي مع الAPI. لا يحتاج الى مكتبة
كيفية الاستخدام	يتم استخدام ال Services عن طريق انشاء كلاس من نوع Service أو IntentService و اضافتها الى ملف AndroidManifest

مثال	Java Code:
	
	public class SyncIntentService extends IntentService {
	@Override
	protected void onHandleIntent(@Nullable Intent intent) {
	// يتم هنا كتابة الكود المراد تشغيلها في الخلفية
	}
	}
	
	AndroidManifest.xml:
	
	<service
	android:name=".data.sync.SyncIntentService"
	android:enabled="true"
	android:exported="false" />
	
	Start a Service:
	
	Intent intent = new Intent(context, SyncIntentService.class);
	context.startService(intent);
	
	Stop a Service:
	1. stopSelt()
	2. context.stopService()

يوجد عدة تطبيقات للServices من بينها: - تشغيل الموسيقى في الخلفية - تزمين البيانات المخزنة في الجهاز مع البيانات الموجودة على الخادم - استقبال الاشعارات	سيناريوهات الاستخدام

مفهوم Job Scheduler

متطلبات التشغيل	نسخ الأندرويد API 21
رابط المكتبة على Gradle	موجود بشكل افتراضي مع جميع الAPIs الأكبر من API 21.
كيفية الاستخدام	يتم استخدام ال Job Scheduler في جدولة المهام في الخلفية. مثلا اذا أردت أن تقوم بتشغيل كود في وقت معين من اليوم و بشرط توفر اتصال انترنت. يكون استخدام الJob Scheduler بتتبع الخطوات الاتية: 1- انشاء الJob Service الذي يحتوي الكود المراد تشغيله في الخلفية 2- انشاء الJob Info الذي يحتوي على الظروف الازم توفرها لتشغيل الJob Service مثل توفر الانترنت، وجود الجهاز على الشاحن، الخ 3- تحديد وقت تشغيل الJob Service باستعمال كلاس JobScheduler

مثال	Java Code:
	
	`public class SyncJobService extends JobService {`
	`  private static final String TAG = "SyncService";`
	
	`  @Override`
	`  public boolean onStartJob(JobParameters params) {`
	`  // يتم هنا كتابة الكود المراد تشغيلها في الخلفية`
	`  return true;`
	`  }`
	
	`  @Override`
	`  public boolean onStopJob(JobParameters params) {`
	`  return true;`
	`  }`
	
	`}`
	AndroidManifest.xml:
	
	`<service`
	`  android:name=".SyncJobService"`
	`  android:label="Sync service"`

	android:permission="android.permission.BIND_JOB_SERVICE" > </service> Scheduling a Service: JobScheduler jobScheduler = (JobScheduler) getSystemService(Context.JOB_SCHEDULER_SERVICE); jobScheduler.schedule(new JobInfo.Builder(SYNC_JOB_ID, new ComponentName(this, SyncJobService.class)) .setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY) .build());
سيناريوهات الاستخدام	تستعمل ال Job Scheduler عامة لجدولة المهام في الخلفية، أحد أكثر السيناريوهات استعمالا هو تنظيم تشغيل خدمة مزامنة البيانات في أوقات معينة و تحت شروط خاصة

مفهوم Work Manager

متطلبات التشغيل	نسخ الأندرويد API 14
رابط المكتبة على Gradle	`implementation "androidx.work:work-runtime:$work_version"`
كيفية الاستخدام	"مثله مثل ال Job Service يتم استخدام ال Work Manager في جدولة المهام في الخلفية. يكون استخدام ال Work Manager بتتبع الخطوات الاتية: 1- انشاء ال Worker الذي يحتوي الكود المراد تشغيله في الخلفية 2- انشاء ال WorkRequest لتحديد وقت تشغيل ال Work و كذالك نوعه مثلا: تشغيل لمرة واحدة فقط أو تشغيل تكراري عند وقت معين 3- اضافة ال Constraints أي شروط تشغيل الWorker 4- جدولة الWork باستعمال كلاس ال WorkManager"

مثال	Java Code:
	
	`public class UploadWorker extends Worker {`
	
	`public UploadWorker(`
	`@NonNull Context context,`
	`@NonNull WorkerParameters params) {`
	`super(context, params);`
	`}`
	
	`@Override`
	`public Result doWork() {`
	`// يتم هنا كتابة الكود المراد تشغيلها في الخلفية`
	`return Result.success()`
	`}`
	`}`
	
	Scheduling a Work:
	
	`Constraints constraints = new Constraints.Builder()`

	.setRequiredNetworkType(NetworkType.CONNECTED) .build(); OneTimeWorkRequest uploadWorkRequest = new OneTimeWorkRequest.Builder(UploadWorker.class) .setConstraints(constraints) .build(); WorkManager.getInstance().enqueue(uploadWorkRequest);
سيناريوهات الاستخدام	نفس استعمالات ال Job Services