Nama : M Rezky Raya Kilwouw

NIM : 222313190

Kelas : 3SI1

PRAKTIKUM 13

PEMROGRAMAN PLATFORM KHUSUS

LAPORAN PRAKTIKUM: BACKGROUND WORK WITH WORKMANAGER

1. PERSIAPAN LINGKUNGAN PENGEMBANGAN

Tahap awal dimulai dengan penambahan dependensi pustaka WorkManager ke dalam proyek agar fungsionalitas penjadwalan tugas latar belakang dapat digunakan.

Penambahan Dependensi pada app/build.gradle.kts: Kode berikut ditambahkan ke dalam blok dependencies untuk mengunduh pustaka work-runtime-ktx:

```
dependencies {
    // Dependensi WorkManager
    implementation("androidx.work:work-runtime-ktx:2.8.1")
}
```

Setelah penambahan kode, proses sinkronisasi Gradle (*Sync Now*) dilakukan untuk menerapkan perubahan.

2. PEMBUATAN KELAS WORKER

Langkah selanjutnya adalah mendefinisikan pekerjaan (task) yang akan dilakukan. Dalam kasus ini, dibuat kelas BlurWorker yang bertugas memburamkan gambar.

File: workers/BlurWorker.kt Kelas ini mewarisi CoroutineWorker untuk mendukung pemrosesan asinkron yang aman. Metode doWork() di-override untuk menyimpan logika utama:

1. Menampilkan notifikasi status.
2. Membaca gambar bitmap.
3. Memproses gambar (efek blur).
4. Menulis hasil ke file temporer.

```
private const val TAG = "BlurWorker"
```

```
class BlurWorker(ctx: Context, params: WorkerParameters) : CoroutineWorker(ctx, params) {

    override suspend fun doWork(): Result {
        val appContext = applicationContext

        // Menampilkan notifikasi bahwa proses sedang berjalan
        makeStatusNotification("Blurring image", appContext)

        return try {
            // Mengambil gambar dari resource (Hardcoded awal)
            val picture = BitmapFactory.decodeResource(
                appContext.resources,
                R.drawable.android_cupcake
            )

            // Memanggil fungsi utilitas untuk memburamkan gambar
            val output = blurBitmap(picture, appContext)

            // Menulis bitmap hasil blur ke file temporer
            val outputUri = writeBitmapToFile(appContext, output)

            makeStatusNotification("Output is $outputUri", appContext)

            // Mengindikasikan pekerjaan sukses
            Result.success()
        } catch (throwable: Throwable) {
            Log.e(TAG, "Error applying blur")
            Result.failure()
        }
```

```
    }
}
```

3. INISIALISASI WORKMANAGER DI VIEWMODEL

Setelah BlurWorker dibuat, WorkManager diinstansiasi di dalam BlurViewModel untuk menjadwalkan pekerjaan tersebut.

File: BlurViewModel.kt Variabel instansi workManager dideklarasikan menggunakan WorkManager.getInstance(application).

```
class BlurViewModel(application: Application) : AndroidViewModel(application) {

    private val workManager = WorkManager.getInstance(application)

}
```

4. PEMBUATAN WORKREQUEST

Untuk menjalankan BlurWorker, sebuah WorkRequest harus dibuat. Implementasi menggunakan OneTimeWorkRequest karena proses ini hanya perlu dijalankan satu kali per klik tombol.

File: BlurViewModel.kt Fungsi applyBlur() diperbarui untuk membuat permintaan kerja dan memasukkannya ke antrean (enqueue).

```
internal fun applyBlur(blurLevel: Int) {
    // Membuat WorkRequest untuk BlurWorker
    val blurRequest = OneTimeWorkRequestBuilder<BlurWorker>()
        .build()

    // Menjadwalkan permintaan ke WorkManager
    workManager.enqueue(blurRequest)
}
```

5. PENAMBAHAN INPUT DAN OUTPUT DATA

Pada tahap sebelumnya, gambar yang diproses bersifat statis. Tahap ini menambahkan fleksibilitas agar Worker dapat memproses URI gambar yang dipilih pengguna melalui objek Data.

Langkah 5.1: Mengirim Data Input (Di ViewModel) Objek Data.Builder digunakan untuk membungkus URI gambar dengan kunci KEY_IMAGE_URI.

```
private fun createInputDataForUri(): Data {
    val builder = Data.Builder()
    imageUri?.let {
        builder.putString(KEY_IMAGE_URI, it.toString())
    }
    return builder.build()
}

internal fun applyBlur(blurLevel: Int) {
    val blurRequest = OneTimeWorkRequestBuilder<BlurWorker>()
        .setInputData(createInputDataForUri()) // Data dikirim ke sini
        .build()

    workManager.enqueue(blurRequest)
}
```

Langkah 5.2: Menerima Input dan Mengirim Output (Di Worker) BlurWorker diperbarui untuk membaca URI dari inputData dan mengirimkan URI hasil melalui Result.success().

```
override suspend fun doWork(): Result {
    val appContext = applicationContext

    // Membaca input URI
    val resourceUri = inputData.getString(KEY_IMAGE_URI)

    return try {
        // Validasi input
        if (TextUtils.isEmpty(resourceUri)) {
            Log.e(TAG, "Invalid input uri")
```

```
            throw IllegalArgumentException("Invalid input uri")
        }

        val resolver = appContext.contentResolver
        val picture = BitmapFactory.decodeStream(
            resolver.openInputStream(Uri.parse(resourceUri))
        )

        val output = blurBitmap(picture, appContext)
        val outputUri = writeBitmapToFile(appContext, output)

        // Mengemas URI output ke dalam Data object
        val outputData = workDataOf(KEY_IMAGE_URI to outputUri.toString())

        // Mengembalikan result success beserta data output
        Result.success(outputData)
    } catch (throwable: Throwable) {
        Log.e(TAG, "Error applying blur")
        Result.failure()
    }
}
```

6. IMPLEMENTASI CHAINING (RANTAI PEKERJAAN)

Aplikasi memerlukan serangkaian tugas berurutan: **Membersihkan File Lama** -> **Memburamkan Gambar** -> **Menyimpan ke Galeri**. WorkManager memfasilitasi ini melalui fitur *Chaining*.

Dua Worker tambahan diasumsikan telah dibuat: CleanupWorker (hapus file temp) dan SaveImageToFileWorker (simpan output).

File: BlurViewModel.kt Kode diperbarui untuk membangun rantai eksekusi:

```
internal fun applyBlur(blurLevel: Int) {
    // Langkah 1: Mulai rantai dengan CleanupWorker
    var continuation = workManager
```

```
        .beginWith(OneTimeWorkRequest.from(CleanupWorker::class.java))

    // Langkah 2: Tambahkan BlurWorker ke rantai
    val blurBuilder = OneTimeWorkRequestBuilder<BlurWorker>()
    blurBuilder.setInputData(createInputDataForUri())
    continuation = continuation.then(blurBuilder.build())

    // Langkah 3: Tambahkan SaveImageToFileWorker ke rantai
    val save = OneTimeWorkRequestBuilder<SaveImageToFileWorker>()
        .build()
    continuation = continuation.then(save)

    // Eksekusi seluruh rantai
    continuation.enqueue()
}
```

7. MENJAMIN PEKERJAAN UNIK (UNIQUE WORK)

Untuk mencegah penumpukan tugas jika pengguna menekan tombol berkali-kali, metode beginWith diganti menjadi beginUniqueWork. Ini memastikan hanya satu rantai pekerjaan manipulasi gambar yang berjalan pada satu waktu.

File: BlurViewModel.kt

```
internal fun applyBlur(blurLevel: Int) {
    // Menggunakan beginUniqueWork dengan kebijakan REPLACE
    // Jika ada pekerjaan berjalan, hentikan dan ganti dengan yang baru
    var continuation = workManager
        .beginUniqueWork(
            IMAGE_MANIPULATION_WORK_NAME, // Nama unik pekerjaan
            ExistingWorkPolicy.REPLACE,
            OneTimeWorkRequest.from(CleanupWorker::class.java)
        )
```

```
    val blurBuilder = OneTimeWorkRequestBuilder<BlurWorker>()
    blurBuilder.setInputData(createInputDataForUri())
    continuation = continuation.then(blurBuilder.build())

    val save = OneTimeWorkRequestBuilder<SaveImageToFileWorker>()
        .addTag(TAG_OUTPUT) // Menambahkan Tag untuk identifikasi
        .build()
    continuation = continuation.then(save)

    continuation.enqueue()
}
```

8. PEMANTAUAN STATUS PEKERJAAN (TAGGING & OBSERVING)

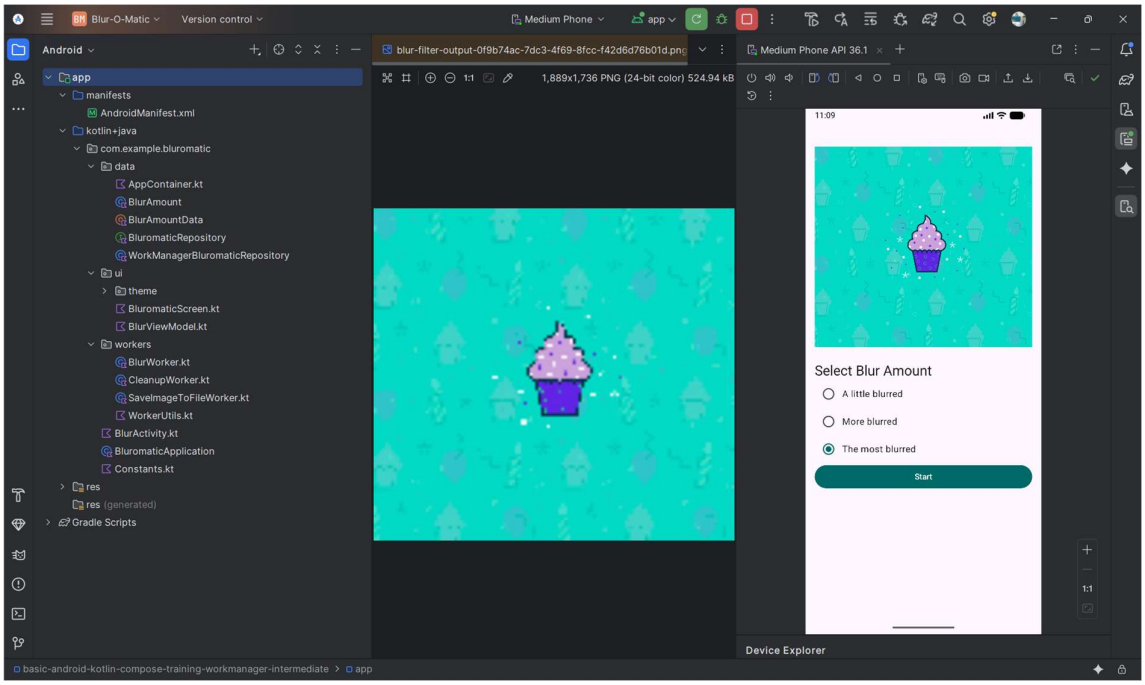
Agar UI dapat bereaksi terhadap status pekerjaan (misalnya menampilkan *progress bar* atau tombol "Cancel"), Worker terakhir (SaveImageToFileWorker) diberi tag TAG_OUTPUT.

Di dalam BlurViewModel, status pekerjaan dipantau menggunakan LiveData.

File: BlurViewModel.kt

```
// Variabel untuk menampung status pekerjaan
val outputWorkInfos: LiveData<List<WorkInfo>>

init {
    // Mengamati WorkInfo berdasarkan Tag "OUTPUT"
    outputWorkInfos = workManager.getWorkInfosByTagLiveData(TAG_OUTPUT)
}
```

Dengan konfigurasi ini, UI aplikasi dapat secara reaktif menampilkan tombol "Download" saat status berubah menjadi SUCCEEDED, atau menampilkan indikator pemuatan saat status RUNNING.



LAPORAN PRAKTIKUM: PROJECT WATER ME! APP (WORKMANAGER)

1. PERSIAPAN PROYEK DAN DEPENDENSI

Proyek dimulai dengan mengunduh kode awal (starter code) yang telah disediakan. Aplikasi ini bertujuan untuk menjadwalkan pengingat penyiraman tanaman menggunakan **WorkManager**.

Langkah pertama memastikan dependensi work-runtime-ktx telah tersedia pada file build.gradle.kts (Module: app) untuk memungkinkan penggunaan fitur penjadwalan tugas latar belakang.

File: app/build.gradle.kts

```
dependencies {
    implementation("androidx.work:work-runtime-ktx:2.8.1")
    // Dependensi lainnya...
}
```

2. IMPLEMENTASI WORKER (WaterReminderWorker)

Untuk menangani tugas latar belakang (background task) berupa pemunculan notifikasi, dibuat sebuah kelas Worker bernama WaterReminderWorker. Kelas ini mewarisi CoroutineWorker agar dapat berjalan secara efisien di latar belakang.

File: worker/WaterReminderWorker.kt Pada kelas ini, fungsi doWork() diimplementasikan untuk melakukan tugas berikut:

1. Mengambil nama tanaman (plantName) dari data input.
2. Memanggil fungsi utilitas makePlantReminderNotification untuk memicu notifikasi sistem.
3. Mengembalikan status Result.success() jika berhasil.

```
class WaterReminderWorker(
    context: Context,
    workerParams: WorkerParameters
) : CoroutineWorker(context, workerParams) {

    override suspend fun doWork(): Result {
        // Mengambil nama tanaman dari input data yang dikirim oleh WorkRequest
        val plantName = inputData.getString(WaterViewModel.PLANT_NAME)

        // Memicu notifikasi menggunakan fungsi utilitas yang telah disediakan
        // makePlantReminderNotification adalah fungsi ekstensi di WorkerUtils.kt
        if (plantName != null) {
            makePlantReminderNotification(plantName, applicationContext)
        } else {
            // Fallback jika nama tanaman tidak ditemukan (opsional)
            makePlantReminderNotification("Plant", applicationContext)
        }

        return Result.success()
    }
}
```

3. IMPLEMENTASI VIEWMODEL DAN PENJADWALAN

Logika utama penjadwalan pengingat diletakkan di dalam WaterViewModel. Di sini, instansi WorkManager diinisialisasi dan fungsi untuk membuat WorkRequest didefinisikan.

File: ui/WaterViewModel.kt

A. Inisialisasi WorkManager

Variabel workManager dideklarasikan menggunakan konteks aplikasi.

```
class WaterViewModel(application: Application) : AndroidViewModel(application) {
    private val workManager = WorkManager.getInstance(application)

    internal companion object {
        const val PLANT_NAME = "plant_name"
    }

    // ...
}
```

B. Membuat Fungsi Penjadwalan (scheduleReminder)

Fungsi scheduleReminder dibuat untuk menerima parameter durasi (duration), satuan waktu (unit), dan nama tanaman (plantName). Fungsi ini bertanggung jawab untuk merakit OneTimeWorkRequest.

Langkah-langkah dalam fungsi ini:

1. **Membuat Data Input**: Mengemas plantName ke dalam objek Data agar bisa dibaca oleh Worker.
2. **Membangun WorkRequest**:
    - Menggunakan OneTimeWorkRequestBuilder.
    - Menetapkan waktu tunda awal menggunakan setInitialDelay.
    - Menyematkan data input menggunakan setInputData.
3. **Menjalankan (Enqueue)**: Mengirim permintaan ke WorkManager.

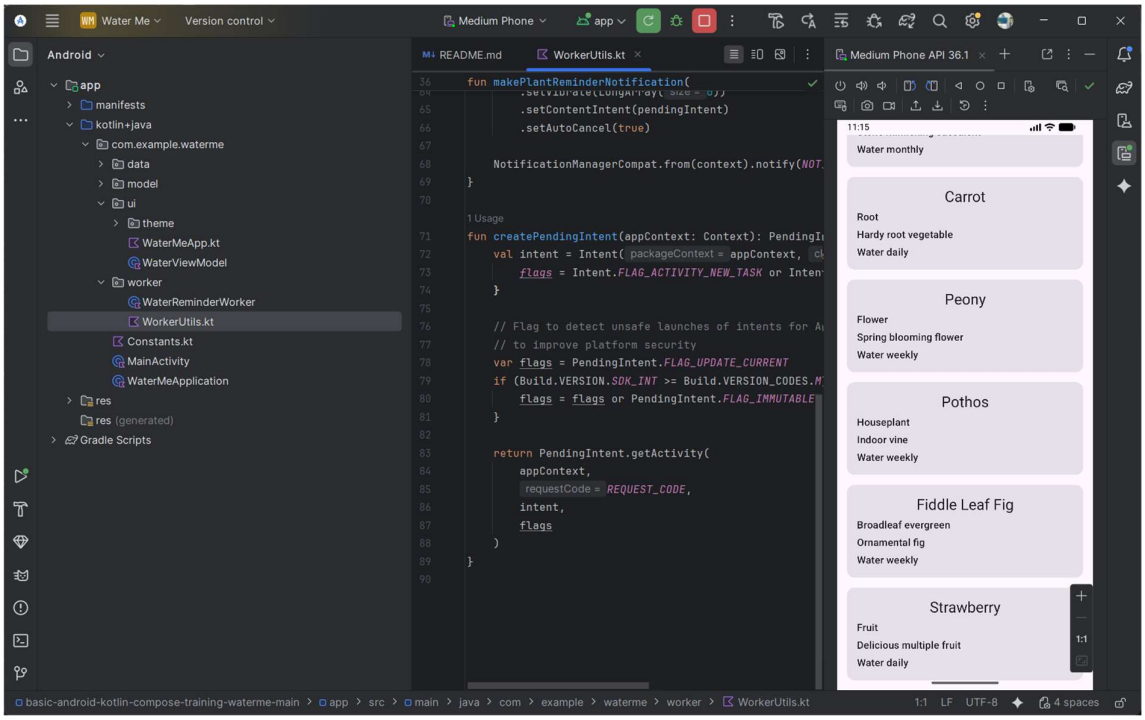
```
internal fun scheduleReminder(
    duration: Long,
    unit: TimeUnit,
    plantName: String
) {
    // 1. Membuat objek Data untuk mengirim nama tanaman ke Worker
    val data = Data.Builder()
        .putString(PLANT_NAME, plantName)
```

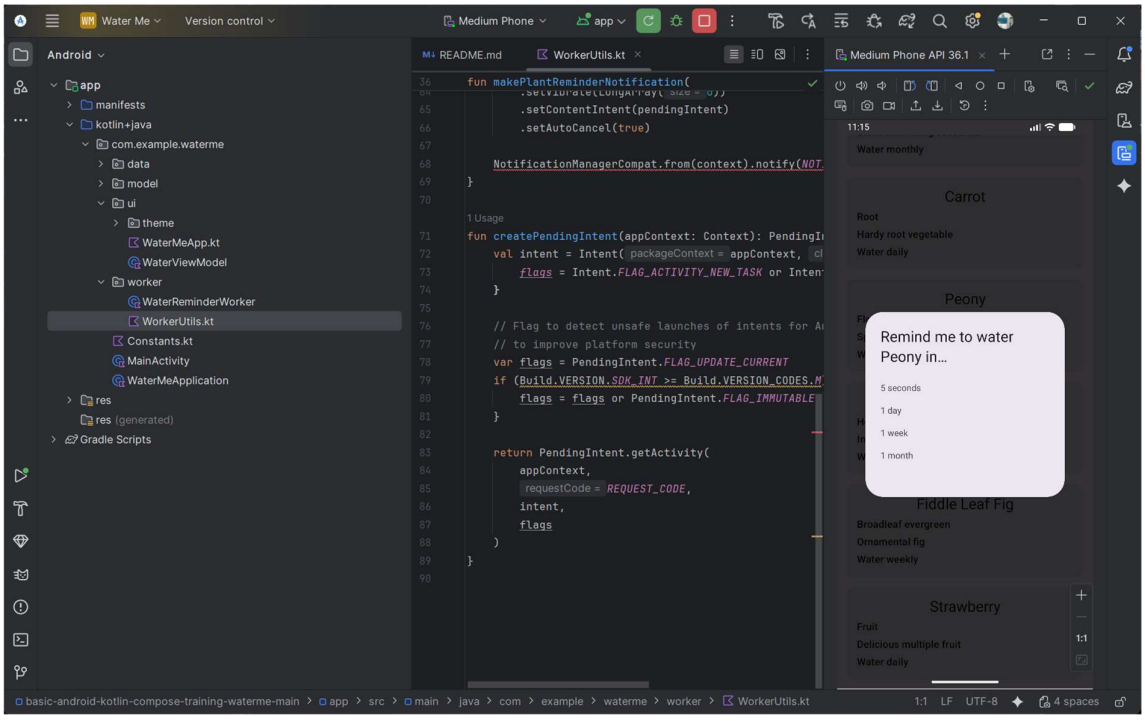
```
        .build()


    // 2. Membangun OneTimeWorkRequest dengan delay
    val workRequest = OneTimeWorkRequestBuilder<WaterReminderWorker>()
        .setInitialDelay(duration, unit) // Kunci utama: Menunda eksekusi sesuai input user
        .setInputData(data)
        .build()


    // 3. Menjadwalkan pekerjaan
    // Menggunakan enqueueUniqueWork agar tidak terjadi duplikasi jika diperlukan,
    // atau cukup enqueue() untuk latihan dasar.
    // Di sini digunakan mekanisme unik berdasarkan nama tanaman agar pengingat bisa dikelola per tanaman.
    workManager.enqueueUniqueWork(
        "$plantName $duration", // Nama unik untuk pekerjaan ini
        ExistingWorkPolicy.REPLACE, // Kebijakan jika pekerjaan sudah ada: Ganti baru
        workRequest
    )
}
```

4. INTEGRASI DENGAN UI

Meskipun logika UI sudah disediakan dalam starter code, fungsi scheduleReminder pada ViewModel dipanggil dari Composable PlantDetailScreen saat pengguna menekan tombol untuk mengatur pengingat.

Kode pada UI mengirimkan parameter waktu (misalnya 5 detik, 1 hari, 30 hari) ke fungsi scheduleReminder, yang kemudian diteruskan ke WorkManager sebagai initialDelay.


Water Me
Version control
Medium Phone
app
Android
README.md
WorkerUtils.kt
Medium Phone API 36.1
app
manifests
kotlin+java
com.example.waterme
data
model
ui
theme
WaterMeApp.kt
WaterViewModel
worker
WaterReminderWorker
WorkerUtils.kt
Constants.kt
MainActivity
WaterMeApplication
res
res (generated)
Gradle Scripts
fun makePlantReminderNotification(
.setContentIntent(pendingIntent)
.setAutoCancel(true)
NotificationManagerCompat.from(context).notify(NOT
}
1 Usage
fun createPendingIntent(appContext: Context): PendingI
val intent = Intent(packageContext = appContext,
flags = Intent.FLAG_ACTIVITY_NEW_TASK or Inten
}
// Flag to detect unsafe launches of intents for A
// to improve platform security
var flags = PendingIntent.FLAG_UPDATE_CURRENT
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
flags = flags or PendingIntent.FLAG_IMMUTABLE
}
return PendingIntent.getActivity(
appContext,
requestCode = REQUEST_CODE,
intent,
flags
)
}
11:15
Water monthly
Carrot
Root
Hardy root vegetable
Water daily
Peony
Flower
Spring blooming flower
Water weekly
Pothos
Houseplant
Indoor vine
Water weekly
Fiddle Leaf Fig
Broadleaf evergreen
Ornamental fig
Water weekly
Strawberry
Fruit
Delicious multiple fruit
Water daily
1:1
basic-android-kotlin-compose-training-waterme-main
app
src
main
java
com
example
waterme
worker
WorkerUtils.kt
1:1
LF
UTF-8
4 spaces


Water Me
Version control
Medium Phone
app
Android
README.md
WorkerUtils.kt
Medium Phone API 36.1
app
manifests
kotlin+java
com.example.waterme
data
model
ui
theme
WaterMeApp.kt
WaterViewModel
worker
WaterReminderWorker
WorkerUtils.kt
Constants.kt
MainActivity
WaterMeApplication
res
res (generated)
Gradle Scripts
fun makePlantReminderNotification(
.setContentIntent(pendingIntent)
.setAutoCancel(true)
NotificationManagerCompat.from(context).notify(NOT
}
1 Usage
fun createPendingIntent(appContext: Context): PendingI
val intent = Intent(packageContext = appContext,
flags = Intent.FLAG_ACTIVITY_NEW_TASK or Inten
}
// Flag to detect unsafe launches of intents for A
// to improve platform security
var flags = PendingIntent.FLAG_UPDATE_CURRENT
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
flags = flags or PendingIntent.FLAG_IMMUTABLE
}
return PendingIntent.getActivity(
appContext,
requestCode = REQUEST_CODE,
intent,
flags
)
}
11:15
Water monthly
Carrot
Root
Hardy root vegetable
Water daily
Peony
Remind me to water Peony in...
5 seconds
1 day
1 week
1 month
Fiddle Leaf Fig
Broadleaf evergreen
Ornamental fig
Water weekly
Strawberry
Fruit
Delicious multiple fruit
Water daily
1:1
basic-android-kotlin-compose-training-waterme-main
app
src
main
java
com
example
waterme
worker
WorkerUtils.kt
1:1
LF
UTF-8
4 spaces