Unidad 7

Rodriguez Segura Fernando

1 Introducción

El proyecto **WorkManager Codelab** es una aplicación educativa desarrollada en Kotlin utilizando Jetpack Compose. Su propósito principal es enseñar a los desarrolladores cómo programar y ejecutar tareas en segundo plano que necesitan una ejecución garantizada, incluso si la aplicación se cierra o el dispositivo se reinicia, utilizando la biblioteca WorkManager.

2 Funcionalidad Principal

La aplicación permite:

- Programar tareas en segundo plano utilizando WorkManager.
- Configurar restricciones para la ejecución de tareas, como batería no baja o conexión a red.
- Encadenar tareas para ejecutar procesos complejos en segundo plano.
- Observar y actualizar el estado de las tareas en la interfaz de usuario utilizando Jetpack Compose.

3 Tecnologías Utilizadas

- **Jetpack Compose**: Toolkit moderno para construir interfaces de usuario en Android.
- **WorkManager**: Biblioteca para programar y ejecutar tareas en segundo plano de manera confiable.
- **Kotlin Coroutines**: Para manejar operaciones asincrónicas de manera eficiente.
- **StateFlow y ViewModel**: Para gestionar el estado y la lógica de la interfaz de usuario.

4 Estructura del Proyecto

- `MainActivity.kt`: Actividad principal que configura la interfaz de usuario y observa el estado de las tareas.
- `BlurWorker.kt`: Clase que define la tarea de desenfocar una imagen utilizando WorkManager.

- `CleanupWorker.kt`: Clase que define la tarea de limpiar archivos temporales.
- `SaveImageToFileWorker.kt`: Clase que define la tarea de guardar la imagen procesada en el almacenamiento.
- `WorkManagerViewModel.kt`: ViewModel que gestiona la lógica de programación y seguimiento de tareas.
- `ui/`: Carpeta que contiene los componentes de la interfaz de usuario construidos con Jetpack Compose.

5 Ejemplo de Código

5.1 Definición de un Worker

Listing 1: Definición de un Worker

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

    override suspend fun doWork(): Result {
        // L gica para desenfocar la imagen
        return Result.success()
    }
}
```

5.2 Programación de Tareas con Restricciones

Listing 2: Programación de Tareas con Restricciones

```
val constraints = Constraints.Builder()
    .setRequiresBatteryNotLow(true)
    .build()

val blurRequest = OneTimeWorkRequestBuilder<BlurWorker>()
    .setConstraints(constraints)
    .build()

workManager.enqueue(blurRequest)
```

5.3 Observación del Estado de la Tarea

Listing 3: Observación del Estado de la Tarea

```
workManager.getWorkInfoByIdLiveData(blurRequest.id)
```

```
    .observe(this) { workInfo ->
        if (workInfo != null && workInfo.state.isFinished) {
            // Actualizar la interfaz de usuario
        }
    }
```

6 Introducción

El proyecto **WorkManager Codelab** es una aplicación educativa desarrollada en Kotlin utilizando Jetpack Compose. Su propósito principal es enseñar a los desarrolladores cómo programar y ejecutar tareas en segundo plano que necesitan una ejecución garantizada, incluso si la aplicación se cierra o el dispositivo se reinicia, utilizando la biblioteca WorkManager.

7 Funcionalidad Principal

La aplicación permite:

- Programar tareas en segundo plano utilizando WorkManager.
- Configurar restricciones para la ejecución de tareas, como batería no baja o conexión a red.
- Encadenar tareas para ejecutar procesos complejos en segundo plano.
- Observar y actualizar el estado de las tareas en la interfaz de usuario utilizando Jetpack Compose.

8 Tecnologías Utilizadas

- **Jetpack Compose**: Toolkit moderno para construir interfaces de usuario en Android.
- **WorkManager**: Biblioteca para programar y ejecutar tareas en segundo plano de manera confiable.
- **Kotlin Coroutines**: Para manejar operaciones asincrónicas de manera eficiente.
- **StateFlow y ViewModel**: Para gestionar el estado y la lógica de la interfaz de usuario.

9 Estructura del Proyecto

- `MainActivity.kt`: Actividad principal que configura la interfaz de usuario y observa el estado de las tareas.
- `BlurWorker.kt`: Clase que define la tarea de desenfocar una imagen utilizando WorkManager.
- `CleanupWorker.kt`: Clase que define la tarea de limpiar archivos temporales.

- `SaveImageToFileWorker.kt`: Clase que define la tarea de guardar la imagen procesada en el almacenamiento.
- `WorkManagerViewModel.kt`: ViewModel que gestiona la lógica de programación y seguimiento de tareas.
- `ui/`: Carpeta que contiene los componentes de la interfaz de usuario construidos con Jetpack Compose.

10 Ejemplo de Código

10.1 Definición de un Worker

Listing 4: Definición de un Worker

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

    override suspend fun doWork(): Result {
        // L gica para desenfocar la imagen
        return Result.success()
    }
}
```

10.2 Programación de Tareas con Restricciones

Listing 5: Programación de Tareas con Restricciones

```
val constraints = Constraints.Builder()
    .setRequiresBatteryNotLow(true)
    .build()

val blurRequest = OneTimeWorkRequestBuilder<BlurWorker>()
    .setConstraints(constraints)
    .build()

workManager.enqueue(blurRequest)
```

10.3 Observación del Estado de la Tarea

Listing 6: Observación del Estado de la Tarea

```
workManager.getWorkInfoByIdLiveData(blurRequest.id)
    .observe(this) { workInfo ->
        if (workInfo != null && workInfo.state.isFinished) {
```

```
            // Actualizar la interfaz de usuario
        }
    }
```

11 Introducción

El proyecto **Water Me! App** es una aplicación educativa desarrollada en Kotlin utilizando Jetpack Compose. Su propósito principal es enseñar a los desarrolladores cómo programar y ejecutar tareas en segundo plano que necesitan una ejecución garantizada, incluso si la aplicación se cierra o el dispositivo se reinicia, utilizando la biblioteca WorkManager.

12 Funcionalidad Principal

La aplicación permite:

- Programar recordatorios de riego para diferentes plantas.
- Configurar restricciones para la ejecución de tareas, como batería no baja o conexión a red.
- Encadenar tareas para ejecutar procesos complejos en segundo plano.
- Observar y actualizar el estado de las tareas en la interfaz de usuario utilizando Jetpack Compose.

13 Tecnologías Utilizadas

- **Jetpack Compose**: Toolkit moderno para construir interfaces de usuario en Android.
- **WorkManager**: Biblioteca para programar y ejecutar tareas en segundo plano de manera confiable.
- **Kotlin Coroutines**: Para manejar operaciones asincrónicas de manera eficiente.
- **StateFlow y ViewModel**: Para gestionar el estado y la lógica de la interfaz de usuario.

14 Estructura del Proyecto

- `MainActivity.kt`: Actividad principal que configura la interfaz de usuario y observa el estado de las tareas.
- `WaterReminderWorker.kt`: Clase que define la tarea de enviar recordatorios de riego utilizando WorkManager.
- `Plant.kt`: Modelo de datos que representa una planta con su información relevante.
- `WaterMeViewModel.kt`: ViewModel que gestiona la lógica de programación y seguimiento de tareas.
- `ui/`: Carpeta que contiene los componentes de la interfaz de usuario construidos con Jetpack Compose.

15 Ejemplo de Código

15.1 Definición de un Worker

Listing 7: Definición de un Worker

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

    override suspend fun doWork(): Result {
        // L gica para enviar el recordatorio de riego
        return Result.success()
    }
}
```

15.2 Programación de Tareas con Restricciones

Listing 8: Programación de Tareas con Restricciones

```
val constraints = Constraints.Builder()
    .setRequiresBatteryNotLow(true)
    .build()

val waterRequest = OneTimeWorkRequestBuilder<WaterReminderWorker>()
    .setConstraints(constraints)
    .build()

workManager.enqueue(waterRequest)
```

15.3 Observación del Estado de la Tarea

Listing 9: Observación del Estado de la Tarea

```
workManager.getWorkInfoByIdLiveData(waterRequest.id)
    .observe(this) { workInfo ->
        if (workInfo != null && workInfo.state.isFinished) {
            // Actualizar la interfaz de usuario
        }
    }
```

16 Conclusión

El proyecto Water Me! App es una excelente referencia para aprender a programar y ejecutar tareas en segundo plano en aplicaciones Android modernas utilizando Kotlin y Jetpack Compose.

Además, demuestra cómo manejar restricciones y observar el estado de las tareas para mantener una interfaz de usuario reactiva y actualizada.