Predavanje br. 13

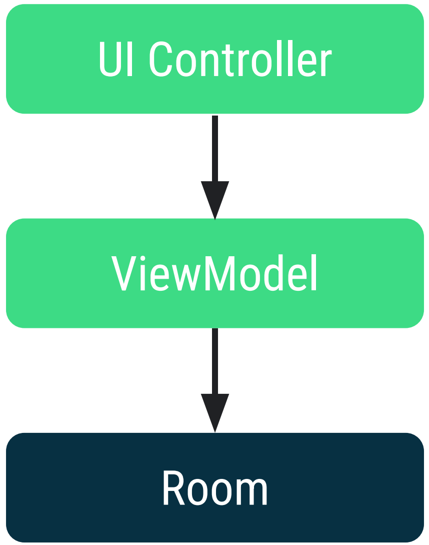
About this lesson

Lesson 12: Repository pattern and `WorkManager`

- Repository pattern
- WorkManager
- Work input and output
- WorkRequest constraints
- Summary

Repository pattern

Existing app architecture



Relative data speeds

Operation	**Relative speed**
Reading from `LiveData`	FAST
Reading from `Room` database	SLOW
Reading from network	SLOWEST

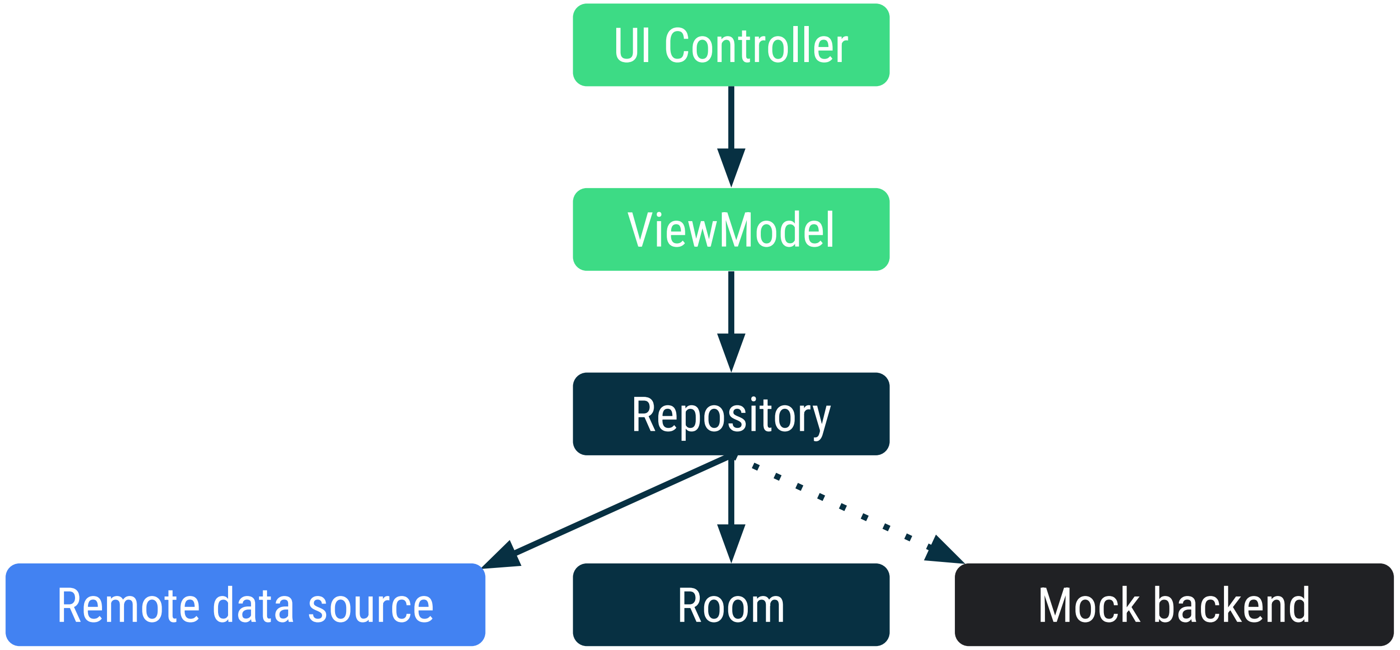
Cache network responses

- Account for long network request times and still be responsive to the user
- Use `Room` locally when retrieval from the network may be costly or difficult
- Can cache based on the least recently used (LRU) value, frequently accessed (FRU) values, or other algorithm

Repository pattern

- Can abstract away multiple data sources from the caller
- Supports fast retrieval using local database while sending network request for data refresh (which can take longer)
- Can test data sources separately from other aspects of your app

App architecture with repository pattern



Implement a repository class

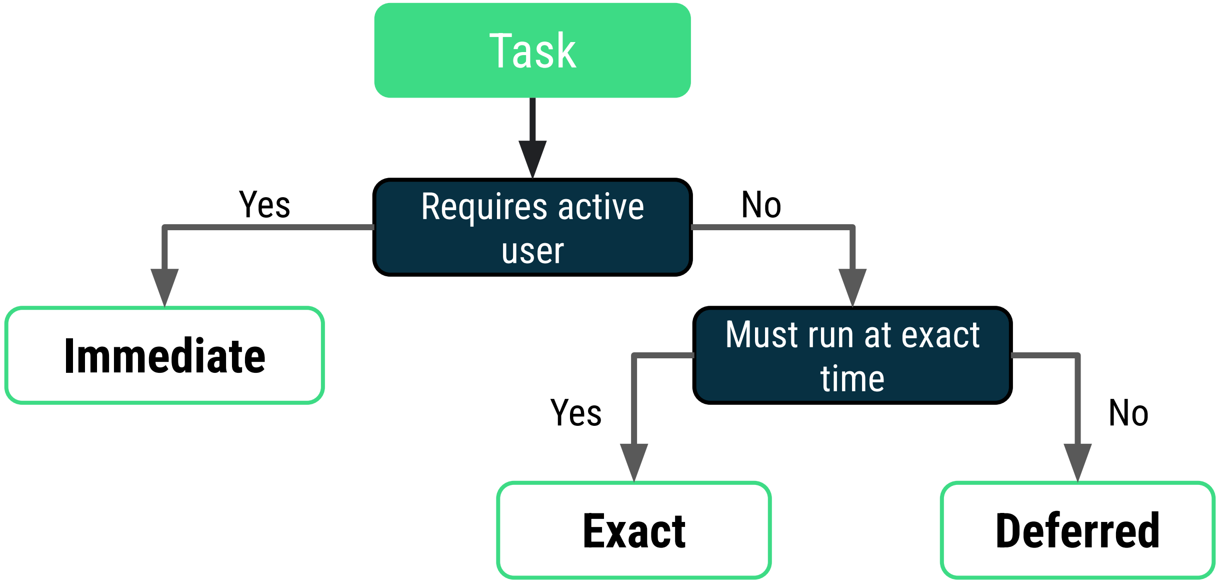
- Provide a common interface to access data:
  - Expose functions to query and modify the underlying data
- Depending on your data sources, the repository can:
  - Hold a reference to the DAO, if your data is in a database
  - Make network requests if you connect to a web service

WorkManager

WorkManager

- Android Jetpack architecture component
- Recommended solution to execute background work (immediate or deferred)
- Opportunistic and guaranteed execution
- Execution can be based on certain conditions

When to use WorkManager



Declare WorkManager dependencies

```
implementation "androidx.work:work-runtime-ktx:$work_version"
```

Important classes to know

- `Worker` - does the work on a background thread, override `doWork()` method
- `WorkRequest` - request to do some work
- `Constraint` - conditions on when the work can run
- `WorkManager` - schedules the `WorkRequest` to be run

Define the work

```
class UploadWorker(appContext: Context, workerParams: WorkerParameters) :
       Worker(appContext, workerParams) {

    override fun doWork(): Result {

       // Do the work here. In this case, upload the images.
       uploadImages()

       // Indicate whether work finished successfully with the Result
       return Result.success()
   }
}
```

Extend CoroutineWorker instead of Worker

```
class UploadWorker(appContext: Context, workerParams: WorkerParameters) :
       CoroutineWorker(appContext, workerParams) {

   override suspend fun doWork(): Result {

       // Do the work here (in this case, upload the images)
       uploadImages()

       // Indicate whether work finished successfully with the Result
       return Result.success()
   }
}
```

WorkRequests

- Can be scheduled to run once or repeatedly
    - `OneTimeWorkRequest`
    - `PeriodicWorkRequest`
- Persisted across device reboots
- Can be chained to run sequentially or in parallel
- Can have constraints under which they will run

Schedule a OneTimeWorkRequest

Create `WorkRequest`:

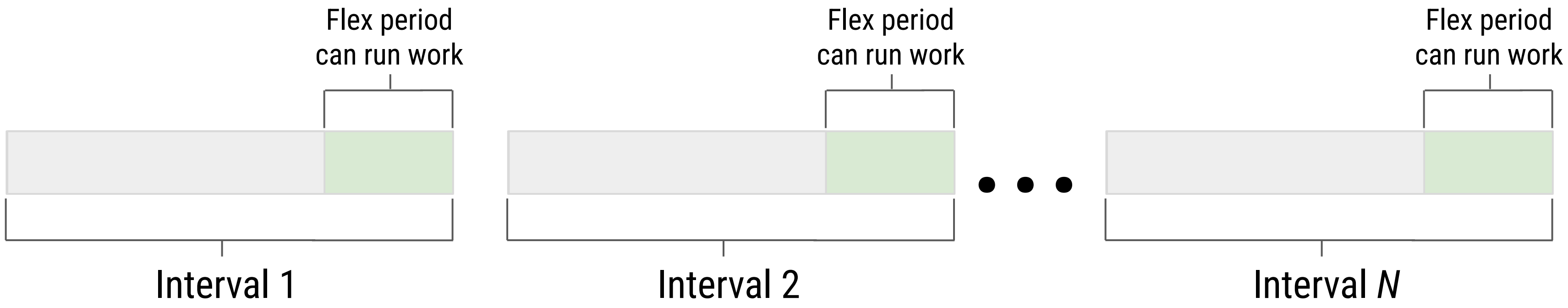
```
val uploadWorkRequest: WorkRequest =
   OneTimeWorkRequestBuilder<UploadWorker>()
       .build()
```

Add the work to the `WorkManager` queue:

```
WorkManager.getInstance(myContext)
    .enqueue(uploadWorkRequest)
```

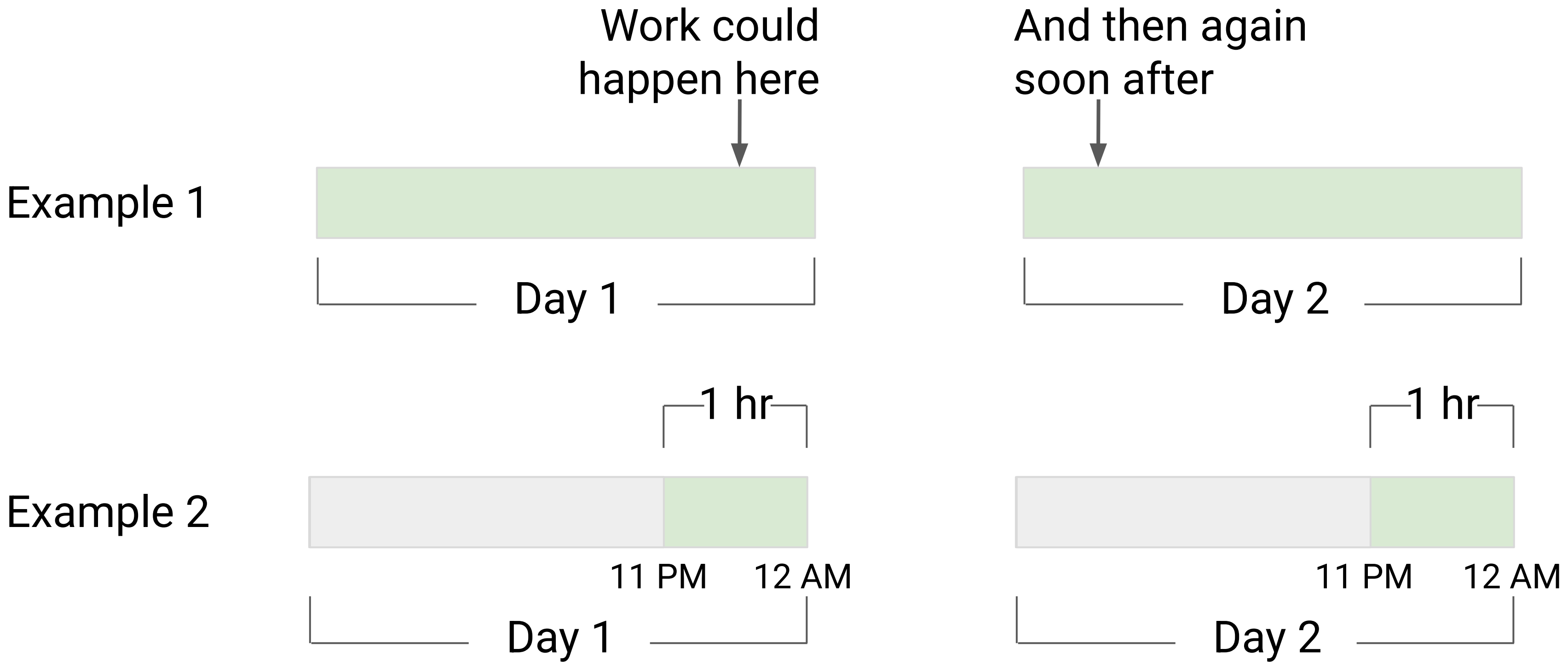
Schedule a PeriodicWorkRequest

- Set a repeat interval
- Set a flex interval (optional)



Specify an interval using `TimeUnit` (e.g., `TimeUnit.HOURS`, `TimeUnit.DAYS`)

Flex interval



PeriodicWorkRequest example

```kotlin
val repeatingRequest = PeriodicWorkRequestBuilder<RefreshDataWorker>(
        1, TimeUnit.HOURS,     // repeatInterval
        15, TimeUnit.MINUTES   // flexInterval
    ).build()
```

Enqueue periodic work

```
WorkManager.getInstance().enqueueUniquePeriodicWork(
     "Unique Name",
     ExistingPeriodicWorkPolicy.KEEP, // or REPLACE
     repeatingRequest
)
```

Work input and output

Define Worker with input and output

```
class MathWorker(context: Context, params: WorkerParameters):
      CoroutineWorker(context, params)  {

    override suspend fun doWork(): Result {
        val x = inputData.getInt(KEY_X_ARG, 0)
        val y = inputData.getInt(KEY_Y_ARG, 0)
        val result = computeMathFunction(x, y)
        val output: Data = workDataOf(KEY_RESULT to result)
        return Result.success(output)
    }
}
```

Result output from doWork()

Result status	Result status with output
`Result.success()`	`Result.success(output)`
`Result.failure()`	`Result.failure(output)`
`Result.retry()`	

Send input data to Worker

```
val complexMathWork = OneTimeWorkRequest<MathWorker>()
    .setInputData(
        workDataOf(
            "X_ARG" to 42,
            "Y_ARG" to 421,
        )
    ).build()

WorkManager.getInstance(myContext).enqueue(complexMathWork)
```

WorkRequest constraints

Constraints

- `setRequiredNetworkType`
- `setRequiresBatteryNotLow`
- `setRequiresCharging`
- `setTriggerContentMaxDelay`
- `requiresDeviceIdle`

Constraints example

```
val constraints = Constraints.Builder()
    .setRequiredNetworkType(NetworkType.UNMETERED)
    .setRequiresCharging(true)
    .setRequiresBatteryNotLow(true)
    .setRequiresDeviceIdle(true)
    .build()

val myWorkRequest: WorkRequest = OneTimeWorkRequestBuilder<MyWork>()
    .setConstraints(constraints)
    .build()
```

Publishing the app

Google Play Store

- zvanična platforma za distribuciju Android aplikacija
- Objava aplikacije znači:
  - da je aplikacija dostupna korisnicima širom svijeta
  - da prolazi kroz tehničku i sigurnosnu provjeru
  - da ima verzionisanje i ažuriranja
  - da autor preuzima odgovornost za sadržaj i podatke
- jedini standardni način distribucije Android aplikacija
- omogućava testiranje, analitiku i monetizaciju

Preduslovi za objavu aplikacije

- Da bi aplikacija bila objavljena, potrebno je:
  - Google Play Developer Account
    - jednokratna registracija (25 USD)
  - Potpisana aplikacija
    - keystore + digitalni potpis
  - Android App Bundle (.aab)
    - obavezni format (APK se više ne koristi)
  - Jedinstveni naziv paketa
    - npr. com.example.myapp

Google Play Console

- U Google Play Console-u se definiše:
  - Osnovne informacije (naziv aplikacije, kratak i dug opis
  - Vizuelni elementi (ikona aplikacije, screenshots)
  - Politike i privatnost
    - privacy policy (obavezno ako se koriste podaci)
    - deklaracija dozvola (kamera, lokacija, internet…)
- Važno:
  - Google posebno kontroliše rad s korisničkim podacima
  - netačne deklaracije → odbijanje aplikacije

Google Play Console

- Track → Release → Review → Publish
- Dostupni track-ovi:
  - Internal testing
    - brzi test (do 100 testera), bez čekanja
  - Closed testing
    - ograničena grupa korisnika
  - Open testing
    - aplikacija dostupna svima koji žele testirati
  - Production
    - javna objava za sve korisnike

Summary

Summary

In Lesson 12, you learned how to:

- Use a repository to abstract the data layer from the rest of the app
- Schedule background tasks efficiently and optimize them using `WorkManager`
- Create custom `Worker` classes to specify the work to be done
- Create and enqueue one-time or periodic work requests

Learn more

- Fetch data
- Schedule tasks with WorkManager
- Define work requests
- Connect ViewModel and the repository
- Use WorkManager for immediate background execution