Universidad Nacional de La Matanza

Persistencia I

Agenda

- Introducción
- Patrón repositorio
- Estado del Arte
- Multiplatform Settings

¿Para qué quiero guardar datos localmente en mi app?

Uso de almacenamiento local

- Para tener datos iniciales cuando se abre por primera vez la app sin conexión.
- Para que las apps sean usables sin conexión.
- Para almacenar datos que el backend no guarda pero pueden ser útiles.
  - Últimas búsquedas
  - Preferencias
  - Archivos accedidos recientemente
  - “Mis posts”
  - Sesión del usuario
  - Como caché de datos offline
  - ...

¿Cómo puedo modelar esto?

¿Cómo hacemos para almacenar y recuperar datos?

Algo muy difundido es el **patrón repositorio**

- Es un patrón de diseño que nos permite encapsular y abstraernos de los detalles de almacenamiento, obtención y búsqueda de objetos
- Relacionado a la **persistencia**
- Se lo puede pensar como una colección de objetos
- Interactúa con objetos de dominio

¿Cómo hacemos para almacenar y recuperar datos?

```
class Book(
    val isbn: String,
    val title: String,
    val author: String
)

interface BooksRepository {

    fun saveBook(book: Book)

    fun findBook(isbn: String): Book?
}
```

```
class MySqlBooksRepository : BooksRepository {
    override fun saveBook(book: Book) {
        // Guardar en base de datos SQL
    }

    override fun findBook(isbn: String): Book? {
        // Obtener de base de datos
    }
}

class FileBooksRepository : BooksRepository {
    override fun saveBook(book: Book) {
        // Guardo un nuevo registro en el archivo
    }

    override fun findBook(isbn: String): Book? {
        // Busco el libro en el archivo
    }
}
```

Estado del arte

Estado del arte

¿Qué opciones tenemos para implementar repositorios?

- SharedPreferences
- DataStore
- SQLite, Room
- Realm

Estado del arte

¿Qué opciones tenemos para implementar repositorios? 

- UserDefaults
- Keychain
- CoreData
- SQLite
- Realm

Estado del arte

SharedPreferences

```
class UserSetting(
    val name: String,
    val value: String
)

interface UserSettingsRepository {

    fun save(setting: UserSetting)

    fun get(name: String): UserSetting?
}
```

```
class SharedPreferencesUserSettingsRepository(
    applicationContext: Context
) : UserSettingsRepository {

    private val sharedPreferences = applicationContext
            .getSharedPreferences("user_settings", Context.MODE_PRIVATE)

    override fun save(setting: UserSetting) {
        sharedPreferences.edit {
            putString(setting.name, setting.value)
        }
    }

    override fun get(name: String): UserSetting? {
        return sharedPreferences.getString(name, null)
            ?.let { value -> UserSetting(name, value) }
    }
}
```

Estado del arte

UserDefaults

```swift
struct UserSetting {
    var name: String
    var value: String
}

protocol UserSettingsRepository {
    func save(setting: UserSetting)

    func get(name: String) -> UserSetting?
}
```

```swift
class UserDefaultsUserSettingsRepository : UserSettingsRepository {

    private let userDefaults: UserDefaults

    init() {
        self.userDefaults = UserDefaults.standard
    }

    func save(setting: UserSetting) {
        userDefaults.set(setting.value, forKey: setting.name)
    }

    func get(name: String) -> UserSetting? {
        if let value = userDefaults.value(forKey: name) as? String {
            return UserSetting(name: name, value: value)
        }
        return nil
    }

}
```

Multiplatform Settings

Multiplatform Settings

- Librería multiplataforma no-oficial pero open source
- Permite almacenar información en formato clave-valor
- Soporte para Android, iOS, Java, Javascript
- Es posible elegir qué implementación usa cada plataforma

Multiplatform Settings

- Integración

```
commonMain {
    dependencies {
        implementation("com.russhwolf:multiplatform-settings:1.0.0-RC")
    }
}
```

Multiplatform Settings

- Uso

  Existe una interfaz `Settings` que tiene distintas variantes de acuerdo a la implementación que se quiera usar

  Para ello, en `commonMain` definimos:

```
expect val settings: Settings

// o sino también

expect fun createSettings(): Settings
```

Multiplatform Settings

- En Android (androidMain), ejemplo usando SharedPreferences

```
actual fun createSettings() : Settings {
    val sharedPrefs: SharedPreferences // obtener preferencias
    return SharedPreferencesSettings(sharedPrefs)
}
```

- En iOS (iosMain), ejemplo usando UserDefaults

```
actual fun createSettings() : Settings {
    val delegate: NSUserDefaults // obtener preferencias
    return NSUserDefaultsSettings(delegate)
}
```

Multiplatform Settings

- Para guardar

```
val settings: Settings = Settings()
settings.putString("nickname", "Messi")
settings.putInt("nivel", 10)
settings["notificaciones"] = false
```

- Para leer

```
val settings: Settings = Settings()

val nickname: String? = settings.getStringOrNull("nickname")
val nivel: Int = settings.getInt("nivel", defaultValue = 1)
val notificaciones: Boolean = settings["notificaciones", false]
```

Multiplatform Settings

- Ejemplo

https://github.com/russhwolf/multiplatform-settings

Links

- Codelab Android Patrón repositorio
- Android Shared Preferences
- Android DataStore
- Apple CoreData
- Multiplatform Settings

Jetpack Multiplatform

- Ejemplo de DataStore Multiplatform
- Jetpack Reference Documentation