Consumo de API REST

Actualizar la interfaz de usuario

{desafío} latam_

Construir una aplicación Android que consume un servicio REST actualizando la interfaz de usuario, acorde al lenguaje Kotlin y a la librería Retrofit

- Unidad 1:
  Acceso a datos en Android
- Unidad 2:
  Consumo de API REST
- Unidad 3:
  Testing
- Unidad 4:
  Distribución del aplicativo Android

{desafío} latam_

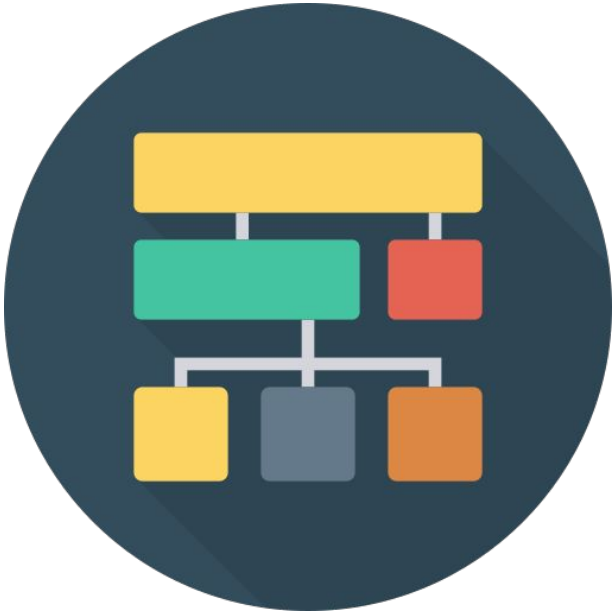
¿Qué aprenderás en esta sesión?

- *Actualizar la interfaz de usuario*

¿Recuerdas qué es StateFlow, cómo crees que se relaciona con Retrofit?

/* Recordando qué es StateFlow */

Recordando qué es StateFlow



StateFlow es una biblioteca basada en el flujo de Kotlin para administrar el estado de la interfaz de usuario en Android.

Proporciona una única fuente de información sobre el estado de su aplicación y ayuda a simplificar la gestión de los cambios de estado de la interfaz de usuario.

Recordando qué es StateFlow

Con StateFlow, puedes rastrear fácilmente el estado de tu aplicación y actualizar la interfaz de usuario en consecuencia, sin tener que administrar manualmente actualizaciones de estado complejas.

Esto da como resultado una base de código más limpia y fácil de mantener y reduce la probabilidad de errores causados por inconsistencias de estado. StateFlow es particularmente útil para administrar flujos de UI complejos y de varios pasos, como los que se encuentran en formularios, flujos de registro y procesos de pago.

/* Actualizar la interfaz de usuario */

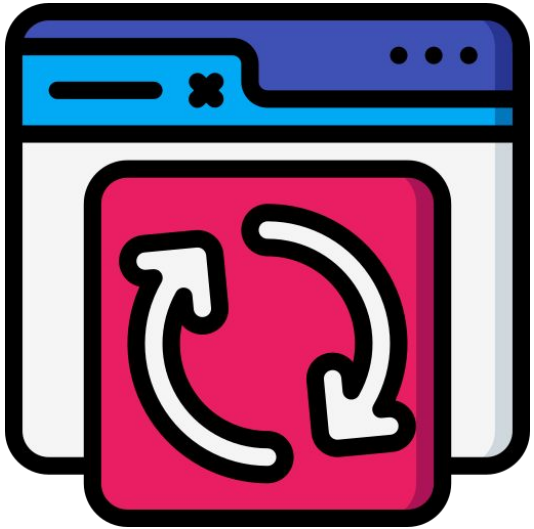
Actualizar la interfaz de usuario

La integración de Retrofit con StateFlow en Android con Kotlin implica la creación de una clase de repositorio que usa Retrofit para obtener datos de un extremo de API y devolver los datos como StateFlow. StateFlow es una API de Kotlin Flows que proporciona un modelo de programación reactivo para administrar datos en su aplicación de Android.

Para integrar Retrofit con StateFlow primero debe crear una interfaz de servicio de Retrofit que defina los puntos finales de la API y el tipo de retorno de los datos. A continuación, crear una clase de repositorio que usa la interfaz de servicio Retrofit para realizar la llamada de red y actualizar los datos como StateFlow. Finalmente, observaría el StateFlow desde su IU utilizando el método de recopilación para actualizar de forma reactiva la IU cuando se emitan nuevos datos.

Actualizar la interfaz de usuario

Cuando el repositorio obtiene datos del extremo de la API mediante Retrofit, actualiza los datos como un StateFlow. Esto permite que la interfaz de usuario se suscriba a StateFlow y se actualice de forma reactiva cuando haya nuevos datos disponibles. Este enfoque reactivo para la administración de datos mediante StateFlow puede ayudar a simplificar su código y mejorar el rendimiento de su aplicación al reducir la cantidad de código repetitivo necesario para manejar los cambios de datos.



Actualizar la interfaz de usuario

Resumiendo:

- Crea una interfaz de servicio Retrofit que defina los endpoints de la API y el tipo de retorno de los datos.
- Crea una clase de repositorio que use la interfaz de servicio Retrofit para realizar la llamada de red y devolver los datos como un StateFlow.
- Observa el StateFlow desde la UI utilizando el método `collect` para actualizar de forma reactiva la interfaz de usuario cuando se emiten nuevos datos.
- Este enfoque simplifica el código y mejora el rendimiento al reducir la cantidad de código repetitivo necesario para manejar los cambios de datos.

Demostración "Retrofit & StateFlow"

Retrofit & StateFlow - GitHub API

Cree una interfaz de servicio Retrofit que define los endpoints de la API y el tipo de retorno de los datos. Usando la API de GitHub la cual devuelve una lista de repositorios como datos JSON. Podemos crear una interfaz de servicio Retrofit para obtener los datos de la siguiente manera:

```
interface GitHubService {
    @GET("users/{username}/repos")
    suspend fun getRepositories(
        @Path("username") username: String
    ): List<Repository>
}
```

Aquí, **Repository** es una clase de datos que representa la respuesta JSON.

Crea una clase de repositorio que use la interfaz de servicio Retrofit para realizar la llamada de red y devolver los datos como un StateFlow. Por ejemplo, creemos una clase `GitHubRepository`:

```
class GitHubRepository {
    private val apiService = Retrofit.Builder()
        .baseUrl("https://api.github.com/")
        .addConverterFactory(GsonConverterFactory.create())
        .build()
        .create(GitHubService::class.java)

    private val _repositories = MutableStateFlow<List<Repository>>(emptyList())
    val repositories: StateFlow<List<Repository>> = _repositories

    suspend fun fetchRepositories(username: String) {
        _repositories.value = apiService.getRepositories(username)
    }
}
```

Aquí, usamos MutableStateFlow para crear un StateFlow mutable para la lista de repositorios. El método fetchRepositories() utiliza apiService para realizar la llamada de red y actualizar el StateFlow de _repositories.

Crea una clase ViewModel que use el repositorio para obtener los datos y exponerlos como un StateFlow a la interfaz de usuario. Por ejemplo, creemos una clase `RepositoriesViewModel`:

```
class RepositoriesViewModel : ViewModel() {
    private val gitHubRepository = GitHubRepository()

    private val _repositories = MutableStateFlow<List<Repository>>(emptyList())
    val repositories: StateFlow<List<Repository>> = _repositories

    fun fetchRepositories(username: String) {
        viewModelScope.launch {
            gitHubRepository.fetchRepositories(username)
        }
    }
}
```

Aquí, creamos un `MutableStateFlow` para la lista de repositorios y lo exponemos como un StateFlow inmutable a la interfaz de usuario. El método `fetchRepositories()` inicia una rutina para obtener los datos del `gitHubRepository`.

Observe el StateFlow desde la interfaz de usuario utilizando el método de recopilación para actualizar de forma reactiva la interfaz de usuario cuando se emiten nuevos datos. Por ejemplo, creemos una `RepositoriesActivity` que muestre la lista de repositorios en un RecyclerView:

```
class RepositoriesActivity : AppCompatActivity() {
    private lateinit var repositoriesViewModel: RepositoriesViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_repositories)

        repositoriesViewModel =
ViewModelProvider(this).get(RepositoriesViewModel::class.java)

        GlobalScope.launch {
            repositoriesViewModel.fetchRepositories("google")
        }

        GlobalScope.launch(Dispatchers.Main) {
            repositoriesViewModel.repositories.collect { repositories ->
                recyclerView.adapter = RepositoryAdapter(repositories)
            }
        }
    }
}
```

Aquí, creamos un repositorioViewModel usando ViewModelProvider y buscamos los repositorios para el usuario de Google en el método onCreate(). Observamos los repositorios de StateFlow utilizando el método de recopilación para actualizar RecyclerView con la lista de repositorios.

/* Recomendaciones */

Recomendaciones

Aquí hay algunas recomendaciones para integrar Retrofit con StateFlow y MVVM:

- **Usa `suspend functions` con Retrofit**: Retrofit brinda soporte para funciones de suspensión, lo que facilita el manejo de solicitudes de red en un contexto de rutina. Puede usar las funciones de suspensión en la interfaz de servicio y las clases de repositorio para evitar bloquear el hilo principal.

- **Cree una capa de repositorio**: para mantener tu código organizado y mantener la separación de preocupaciones, crea una capa de repositorio entre el modelo de vista y la red. El repositorio puede usar Retrofit para realizar solicitudes de red y exponer la respuesta como StateFlow al modelo de vista.

Recomendaciones

- **Utiliza una única fuente de información**: utiliza una única fuente de información para sus datos almacenándolos en un StateFlow en la capa del repositorio. Esto garantiza que los datos estén siempre actualizados y que la interfaz de usuario se actualice automáticamente cuando cambien los datos.

- **Evita usar LiveData y Flow juntos:** si bien es posible usar LiveData y Flow juntos, puede generar un comportamiento inesperado y hacer que su código sea más complejo. En su lugar, usa **StateFlow** en toda su aplicación para mantener su código consistente y más fácil de mantener.

Recomendaciones

- **Utiliza un enfoque reactivo:** utiliza el método `collect` o `collecLatest` para observar el StateFlow en el modelo de vista y actualice de forma reactiva la interfaz de usuario cuando se emitan nuevos datos. Esto garantiza que la interfaz de usuario esté siempre actualizada con los datos más recientes de la red.

- **Maneja los errores de forma adecuada:** Retrofit puede generar excepciones por varios motivos, como errores de red, errores de tiempo de espera y errores del servidor. Asegúrate de manejar estos errores correctamente y muestra los mensajes de error apropiados al usuario.

Al seguir estas recomendaciones, puedes crear una base de código bien organizada, reactiva y mantenible.

Nombra alguna de las recomendaciones al momento usar Retrofit con StateFlow

{desafío} latam_

Próxima sesión...

- *Guia de ejercicios*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales