# Guía de ejercicios - Consumo de API REST (I)

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

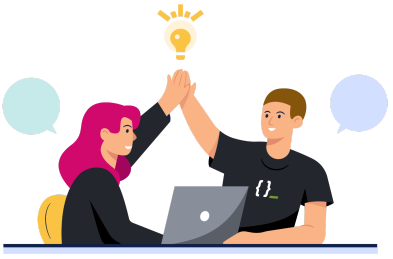
## ¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto en clase.

**¡Vamos con todo!**

## Tabla de contenidos





**¡Comencemos!**

## Actividad guiada: HTTP Request con Kotlin y OkHttp (No extra libraries):

¡Empecemos!

Para realizar una solicitud HTTP en Android con Kotlin usando OkHttp y sin librerías extras, puedes seguir estos pasos:

1. Agrega la dependencia de OkHttp a su proyecto sumando la siguiente línea al archivo build.gradle de su aplicación:

```
implementation("com.squareup.okhttp3:okhttp:4.9.3")
```

2. Crea una clase que represente tu endpoint de API y use OkHttp para realizar la solicitud HTTP. Esta clase debe ser responsable de realizar la solicitud HTTP y devolver la respuesta.

Puedes definirlo así:

```
class ApiClient {
    private val client = OkHttpClient()

    fun makeRequest(url: String): String? {
        val request = Request.Builder()
            .url(url)
            .build()
        try {
            val response = client.newCall(request).execute()
            return response.body?.string()
        } catch (e: IOException) {
            e.printStackTrace()
        }

        return null
    }
}
```

3. Crea una clase de repositorio que será responsable de recuperar datos del extremo de la API. Esta clase de repositorio debe usar la clase ApiClient para realizar la solicitud HTTP y devolver la respuesta a ViewModel.

Puedes definirlo así:

```
class MyRepository {
    private val apiClient = ApiClient()

    private val _data = MutableStateFlow<String?>(null)
    val data: StateFlow<String?> = _data

    suspend fun fetchData() {
        val response = withContext(Dispatchers.IO) {

apiClient.makeRequest("https://your-api-endpoint.com/your-resource")
        }
        _data.value = response
    }
}
```

4. Crea una clase ViewModel que será responsable de obtener los datos del repositorio y proporcionarlos a la Vista. El ViewModel debe desacoplarse del marco de trabajo de Android, por lo que no debe tener dependencias específicas de Android.

Puedes definirlo así:

```
class MyViewModel(private val repository: MyRepository) : ViewModel() {
    val data: StateFlow<String?> = repository.data

    fun fetchData() {
        viewModelScope.launch {
            repository.fetchData()
        }
    }
}
```

5. En tu Actividad o Fragmento, crea una instancia de ViewModel y observa sus datos usando StateFlow.

Puedes definirlo así:

```
class MyActivity : AppCompatActivity() {
    private val viewModel: MyViewModel by viewModels {
        MyViewModelFactory(MyRepository())
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        lifecycleScope.launch {
            viewModel.data.collect { data ->
                // Update the UI with the data
            }
        }

        viewModel.fetchData()
    }
}
```

Ten presente que, en este caso, no estamos manejando ningún tipo de excepción.

## Actividad guiada: HTTP Request con Kotlin, OkHttp y manejo de excepciones (No extra libraries):

En esta Actividad Guiada usaremos el ejemplo anterior, pero esta vez manejaremos las posibles excepciones

1. Cree una clase sellada que represente los diferentes estados posibles de su solicitud de API:

```
sealed class ApiState {
    object Loading : ApiState()
    data class Success(val data: String?) : ApiState()
    data class Error(val message: String?) : ApiState()
}
```

2. Cree una clase ViewModel que será responsable de obtener los datos del repositorio y proporcionarlos a la Vista. ViewModel debe desacoplarse del marco de trabajo de Android, por lo que no debe tener dependencias específicas de Android.

Puedes definirlo así:

```
class MyViewModel(private val repository: MyRepository) : ViewModel() {
    private val _apiState = MutableStateFlow<ApiState>(ApiState.Loading)
    val apiState: StateFlow<ApiState> = _apiState

    fun fetchData() {
        viewModelScope.launch {
            _apiState.value = ApiState.Loading
            try {
                val response = repository.fetchData()
                _apiState.value = ApiState.Success(response)
            } catch (e: IOException) {
                _apiState.value = ApiState.Error("Failed to fetch data
from API endpoint")
            }
        }
    }
}
```

3. En su Fragmento, cree una instancia de ViewModel y observe su `apiState` usando StateFlow.

Puedes definirlo así:

```
class MyFragment : Fragment() {
    private val viewModel: MyViewModel by viewModels {
        MyViewModelFactory(MyRepository())
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_my, container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?)
{
        super.onViewCreated(view, savedInstanceState)

        viewModel.fetchData()

        lifecycleScope.launch {
            viewModel.apiState.collect { apiState ->
                when (apiState) {
                    is ApiState.Loading -> {
                        // Show a loading spinner or other feedback to
the user
                    }
                    is ApiState.Success -> {
                        // Update the UI with the data
                    }
                    is ApiState.Error -> {
                        // Handle the error, e.g. display an error
message to the user
                    }
                }
            }
        }
    }
}
```

Con esta implementación puede manejar fácilmente diferentes estados y excepciones en su Fragmento en función de la respuesta de la solicitud de la API.

## ¡Manos a la obra! - Request con Kotlin y validar resultado usando Postman

Siguiendo los mismos pasos descritos en los ejercicios anteriores, consulta el siguiente endpoint y compara la respuesta consultando el mismo endpoint pero esta vez usando Postman

Endpoint: https://cat-fact.herokuapp.com/facts

## Respuestas:

Respuesta del ejercicio "**Request con Kotlin y validar resultado usando Postman**" json proveniente de "Endpoint: https://cat-fact.herokuapp.com/facts"

JSON:


```
[{"status":{"verified":true,"feedback":"","sentCount":1},"_id":"5887e1d8
5c873e0011036889","user":"5a9ac18c7478810ea6c06381","text":"Cats make
about 100 different sounds. Dogs make only about
10.","__v":0,"source":"user","updatedAt":"2020-09-03T16:39:39.578Z","typ
e":"cat","createdAt":"2018-01-15T21:20:00.003Z","deleted":false,"used":t
rue},{"status":{"verified":true,"sentCount":1},"_id":"588e746706ac2b0011
0e59ff","user":"588e6e8806ac2b00110e59c3","text":"Domestic cats spend
about 70 percent of the day sleeping and 15 percent of the day
grooming.","__v":0,"source":"user","updatedAt":"2020-08-26T20:20:02.359Z
","type":"cat","createdAt":"2018-01-14T21:20:02.750Z","deleted":false,"u
sed":true},{"status":{"verified":true,"sentCount":1},"_id":"58923f2fc387
8c0011784c79","user":"5887e9f65c873e001103688d","text":"I don't know
anything about
cats.","__v":0,"source":"user","updatedAt":"2020-08-23T20:20:01.611Z","t
ype":"cat","createdAt":"2018-02-25T21:20:03.060Z","deleted":false,"used"
:false},{"status":{"verified":true,"sentCount":1},"_id":"5894af975cdc740
0113ef7f9","user":"5a9ac18c7478810ea6c06381","text":"The technical term
for a cat's hairball is a
bezoar.","__v":0,"source":"user","updatedAt":"2020-11-25T21:20:03.895Z",
"type":"cat","createdAt":"2018-02-27T21:20:02.854Z","deleted":false,"use
d":true},{"status":{"verified":true,"sentCount":1},"_id":"58e007cc0aac31
001185ecf5","user":"58e007480aac31001185ecef","text":"Cats are the most
popular pet in the United States: There are 88 million pet cats and 74
million
dogs.","__v":0,"source":"user","updatedAt":"2020-08-23T20:20:01.611Z","t
ype":"cat","createdAt":"2018-03-01T21:20:02.713Z","deleted":false,"used"
:false}]
```


Utiliza Postman o un servicio online como: https://jsonformatter.curiousconcept.com/ para dar formato al Json, también puedes usar alguna herramienta Diff para comparar archivos en casos que los archivos JSON sean demasiado grandes.

## Preguntas de proceso

**Reflexiona:**

- ¿Existe algún contenido que te haya resultado interesante o de utilidad hasta ahora? ¿Por qué?
- ¿Qué ejercicio se te hizo más difícil? ¿Puedes identificar por qué es más dificultoso para ti?
- ¿Cómo crees que podrían hacer más sencillo tu proceso de aprendizaje?


## Preguntas de cierre

- ¿Qué es un cliente HTTP?
- ¿Qué es un error 500?
- ¿El código 201 en HTTP es un error?
- ¿Para qué sirve OkHttp?
- ¿Cuál es la diferencia entre Error y Exception?