Paso 1: Instalación y Configuración

- **No hay código aquí**: Solo instrucciones para descargar e instalar Android Studio. Crea un proyecto base con Compose activado. El proyecto generado incluye archivos como **MainActivity.kt** y **build.gradle** con dependencias mínimas.

Paso 2: Jetpack Compose Básico (Column, Row, Text, Icon)

Este paso introduce Compose. Editas **MainActivity.kt** y **build.gradle**.

- **Dependencias en build.gradle**:
    - **implementation 'androidx.compose.ui:ui:1.5.0'**: Biblioteca core de Compose para UI declarativa.
    - **implementation 'androidx.compose.material3:material3:1.1.0'**: Componentes Material Design 3 (botones, textos, etc.).
    - **implementation 'androidx.compose.ui:ui-tooling-preview:1.5.0'**: Permite previews en Android Studio.
- **En MainActivity.kt**:
    - **package com.example.miprimeraapp**: Define el paquete del proyecto (cambia según tu app).
    - Imports: Traen clases necesarias (ej. **ComponentActivity** para la actividad principal, **setContent** para definir UI en Compose).
    - **class MainActivity : ComponentActivity()**: Clase principal de la app. Extiende **ComponentActivity** (base para apps modernas).
        - **override fun onCreate(savedInstanceState: Bundle?)**: Método que se ejecuta al iniciar la app. Llama a **setContent** para renderizar la UI.
        - **setContent { MiPrimeraAppTheme { Greeting("Mundo") } }**: Define el contenido de la pantalla. **MiPrimeraAppTheme** aplica el tema global (generado por Android Studio). Llama a la función **Greeting**.
    - @**Composable fun Greeting(name: String)**: Función composable (declarativa). Recibe un **name** y dibuja UI.
        - **Column(...)**: Contenedor que apila elementos verticalmente.
            - **modifier = Modifier.fillMaxSize()**: Ocupa toda la pantalla.

- **verticalArrangement = Arrangement.Center**: Centra verticalmente.
- **horizontalAlignment = Alignment.CenterHorizontally**: Centra horizontalmente.

- **Row { ... }**: Dentro de Column, apila horizontalmente.
  - **Icon(...)**: Muestra un ícono (estrella). **imageVector = Icons.Default.Star**: Ícono predeterminado. **contentDescription**: Texto accesible. **tint = Color.Yellow**: Color amarillo.
  - **Text(...)**: Muestra texto. **text = "Hola, $name!"**: Interpolación de string. **style = MaterialTheme.typography.headlineMedium**: Estilo de texto grande.

- **@Preview(showBackground = true) @Composable fun DefaultPreview()**: Función para preview en Android Studio (no afecta la app real). Muestra **Greeting** en el editor.

Qué hace en total: Crea una pantalla centrada con una estrella y "Hola, Mundo!". Ejecuta en emulador.

Paso 3: Crear Listas Dinámicas con LazyColumn

Agregas datos y una lista eficiente.

- **En MainActivity.kt** (arriba de la clase):
  - **data class Item(val title: String, val description: String)**: Modelo de datos simple (como una estructura). Tiene dos campos: título y descripción.
  - **val sampleItems = listOf(...)**: Lista estática de **Item** para probar.
- **Nueva función ItemList**:
  - **@Composable fun ItemList(items: List<Item>)**: Función que recibe una lista y la muestra.
    - **LazyColumn(modifier = Modifier.fillMaxSize())**: Lista vertical eficiente (solo renderiza elementos visibles). Ocupa toda la pantalla.
      - **items(items) { item -> ... }**: Itera sobre la lista. Para cada **item**, crea un **Column** con padding.

        - **Column(modifier = Modifier.padding(16.dp))**: Contenedor con espacio alrededor.
            - **Text(text = item.title, ...)**: Muestra el título con estilo mediano.
            - **Text(text = item.description, ...)**: Muestra la descripción con estilo cuerpo.
- **En setContent**: Cambia a **ItemList(sampleItems)** para mostrar la lista.

Qué hace en total: Muestra una lista desplazable de títulos y descripciones. **LazyColumn** es eficiente para listas grandes.

Paso 4: Conectar a APIs REST con Retrofit y ProgressBar

Integra API, ViewModel y loading.

- **Dependencias en build.gradle**:
    - **retrofit2:retrofit** y **converter-gson**: Para hacer peticiones HTTP y convertir JSON a objetos Kotlin.
    - **lifecycle-viewmodel-compose**: Para usar ViewModels en Compose.
- **Archivo data/Post.kt**:
    - **data class Post(val id: Int, val title: String, val body: String)**: Modelo para datos de la API (id, título, cuerpo).
- **Archivo data/ApiService.kt**:
    - **interface ApiService { @GET("posts") suspend fun getPosts(): List<Post> }**: Define la API. **@GET("posts")**: Endpoint para obtener posts. **suspend**: Función asíncrona (corutinas).
    - **val retrofit = Retrofit.Builder()...**: Configura Retrofit con URL base y conversor Gson.
    - **val apiService = retrofit.create(ApiService::class.java)**: Crea instancia de la API.
- **Archivo viewmodel/PostViewModel.kt**:
    - **class PostViewModel : ViewModel()**: Maneja estado y lógica (no UI).
        - **var posts by mutableStateOf<List<Post>>(emptyList())**: Estado mutable para la lista de posts (inicia vacía).

        - **var isLoading by mutableStateOf(false)**: Estado para mostrar loading.
        - **fun fetchPosts()**: Función para cargar datos.
            - **viewModelScope.launch { ... }**: Ejecuta en corutina (asíncrono).
                - **isLoading = true**: Activa loading.
                - **posts = apiService.getPosts()**: Llama a la API y asigna resultado.
                - **catch (e: Exception)**: Maneja errores (opcional).
                - **isLoading = false**: Desactiva loading.
- **En MainActivity.kt** (actualizaciones):
    - Imports: Agrega **LaunchedEffect**, **viewModel**, etc.
    - En **setContent**:
        - **val viewModel: PostViewModel = viewModel()**: Obtiene instancia del ViewModel.
        - **LaunchedEffect(Unit) { viewModel.fetchPosts() }**: Ejecuta **fetchPosts** una vez al cargar.
        - **if (viewModel.isLoading) { CircularProgressIndicator(...) } else { ItemList(...) }**: Si loading, muestra indicador circular; sino, muestra lista mapeada (**posts.map { Item(it.title, it.body) }** convierte **Post** a **Item**).

Qué hace en total: Carga posts de la API asíncronamente, muestra ProgressBar durante la carga, y lista los datos. Maneja estado con ViewModel.