Retrofit

Menghubungkan Android ke REST API



Pertemuan 11
Mata Kuliah:
Pemrograman Aplikasi Mobile

Dosen:
Mahisa Ajy Kusuma, S.Kom., MMSI.

Informasi Perkuliahan

- **Pertemuan 11** — HTTP Client & REST API dengan Retrofit
- **Durasi:** 3 jam (170 menit materi + istirahat)
- **Peserta:** semester 5, sudah menguasai Activity, Layout XML, Intent, widget dasar

Capaian pembelajaran

Mahasiswa mampu:

- menjelaskan cara kerja REST API
- menguji API dengan **Bruno**
- membuat mock API di **MockAPI.io**
- membangun app CRUD dengan **Retrofit + Gson + Coroutines**

Mengapa Materi Ini Penting

📱

Aplikasi = klien server

Ojek online, marketplace, media sosial, absensi kampus — semuanya mengambil data dari server.

☁️

Data utama ada di server

Aplikasi mengambil dan mengirim data tersebut melalui **API**.

🛠️

Yang akan kita praktikkan

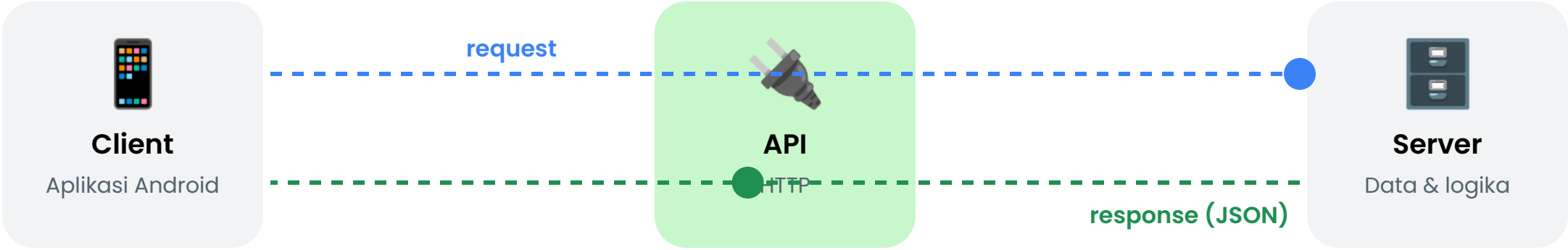
Membuat aplikasi CRUD lengkap yang terhubung ke server melalui internet.

🎯

Keterampilan inti

Ambil data dari API, tampilkan, dan kirim data kembali.

API & Model Client–Server



🍽️ Analogi restoran

Kamu (aplikasi) → **pelayan** (API) → **dapur** (server). Kamu tidak masuk dapur, kamu memesan lewat pelayan.

Platform-agnostic

Komunikasi berbasis teks standar: server yang sama bisa dipakai Android, iOS, dan Web.

REST API

Semua adalah **resource** dengan URL (**endpoint**). Aksi ditentukan oleh **HTTP method**, bukan URL.

Aksi	Method	Endpoint
Ambil semua post	GET	/posts
Ambil satu post	GET	/posts/{id}
Buat post baru	POST	/posts
Ganti seluruh isi post	PUT	/posts/{id}
Ganti sebagian isi post	PATCH	/posts/{id}
Hapus post	DELETE	/posts/{id}

HTTP: Request & Response

📤 Request

- **Method** — GET, POST, …
- **URL** — `…/api/v2/posts/1`
- **Headers** — `Content-Type: application/json`
- **Body** — JSON (POST/PUT/PATCH)

📥 Response

- **Status code** — ringkasan hasil (3 digit)
- **Headers** — metadata balasan
- **Body** — data balasan, biasanya JSON

200 OK
sukses (GET, PUT, PATCH, DELETE)

201 Created
hasil POST

404 Not Found
salah dari client

500 Server Error
salah dari server

2xx sukses · 4xx salah client · 5xx salah server — di Retrofit, 4xx/5xx ditangkap sebagai `HttpException`

PUT vs PATCH

Data di server

```
{
  "title": "Lama",
  "body": "Isi lama"
}
```

PATCH

```
{ "title": "Baru" }
```

→

```
{
  "title": "Baru",
  "body": "Isi lama"
}
```

PUT

```
{ "title": "Baru" }
```

→

```
{
  "title": "Baru",
  "body": ""
}
```

PATCH = sebagian

Field yang tidak dikirim **tetap utuh**.

PUT = seluruhnya

Field yang tidak dikirim **hilang / dikosongkan**.

JSON → Objek Kotlin

```
{
  "title": "Judul postingan",
  "body": "Isi postingan",
  "id": "1"
}
```

→ Gson
deserialisasi

```
data class Post(
    val id: String,
    val title: String,
    val body: String
)
```

```
data class PostRequest(
    val title: String? = null,
    val body: String? = null
)
```

- Nama properti **harus sama persis** dengan kunci JSON
- `Post` = model respons · `PostRequest` = model request
- Field yang `null` tidak ikut dikirim → inilah yang membuat **PATCH bisa mengubah sebagian data saja**

```
PostRequest(title = "Baru")  →  { "title": "Baru" }
// body = null → tidak dikirim → body lama tetap utuh
```

Uji API Dulu dengan Bruno

API client open source, offline-first, collection berupa file teks `.bru` (mudah dibagikan lewat Git). Alternatif: Postman, Insomnia, `curl`.

1

Collection
Create Collection `Pertemuan11`

2

`GET`
Get Posts → **200**, body array JSON

3

`POST`
Body JSON → **201** + `id` baru

4

`PUT`
Kirim title & body → semua berubah

5

`PATCH`
Hanya title → `body` tetap utuh

6

`DELETE`
200, GET ulang → **404**

Jangan koding dulu sebelum tahu bentuk nyata response. **Kalau di Bruno sudah jelas, di Android tinggal "diterjemahkan".**

MockAPI.io: Server Tiruan

REST API palsu tapi **benar-benar berfungsi** — tanpa menulis backend.

	JSONPlaceholder	MockAPI.io
Setup	langsung pakai	akun + resource
Data	statis, tidak disimpan	**CRUD tersimpan**
Cocok untuk	demo GET cepat	praktik CRUD penuh

- New Project → New Resource `posts`
- Field: `title`, `body` (string) → Generate
- Base URL: `https://<kunci>.mockapi.io/api/v2/`

⚠️ `id` di MockAPI bertipe **String**, bukan angka → `val id: String`. POST → 201, PUT/PATCH → 200, DELETE → 200 lalu GET → 404.

Arsitektur Aplikasi

MainActivity — memanggil suspend fun di coroutine
Layar yang meminta data; dijalankan di coroutine agar tampilan tidak membeku.

RetrofitClient — singleton: base URL + OkHttp + Gson
Pengaturan jaringan yang dibuat sekali (alamat server, OkHttp, Gson) lalu dipakai seluruh aplikasi.

ApiService — interface beranotasi (@GET/@POST/…)
Daftar "menu" endpoint; anotasi memberi tahu Retrofit method dan alamat tiap fungsi.

INTERNET → Server MockAPI — response JSON
Request dikirim ke server, lalu server membalas dengan data berformat JSON.

Gson — JSON → objek Kotlin (`Post`)
Penerjemah yang mengubah teks JSON menjadi objek Kotlin yang mudah dipakai.

Adapter / UI — tampil di layar
Objek `Post` dipasang ke daftar sehingga datanya muncul di layar pengguna.

Pembagian package

`data/model/` — Post, PostRequest
`data/api/` — ApiService, RetrofitClient
`root` — Activity & Adapter

Prinsip: kode jaringan dan kode tampilan dipisahkan, supaya lebih mudah dipahami dan diperbaiki.

Retrofit: ApiService

```
interface ApiService {
    @GET("posts")
    suspend fun getPosts(): List<Post>

    @GET("posts/{id}")
    suspend fun getPostById(@Path("id") id: String): Post

    @POST("posts")
    suspend fun createPost(@Body request: PostRequest): Response<Post>

    @PUT("posts/{id}")
    suspend fun updatePost(@Path("id") id: String,
        @Body request: PostRequest): Response<Post>

    @PATCH("posts/{id}")
    suspend fun patchPost(@Path("id") id: String,
        @Body request: PostRequest): Response<Post>

    @DELETE("posts/{id}")
    suspend fun deletePost(@Path("id") id: String): Response<ResponseBody>
}
```

- Library HTTP client **type-safe** dari Square. Kita tulis **apa**, Retrofit mengerjakan **bagaimana**.
- `@Path` menyisipkan nilai ke path; `@Body` dijadikan JSON.
- `Post` / `List<Post>` → kita hanya mendapat **datanya**. Dipakai di GET.
- `Response<Post>` → kita mendapat **data + status code** (200, 201, …). Dipakai di POST, PUT, PATCH.
- DELETE: `Response<ResponseBody>` → balasannya hanya teks mentah data yang terhapus, jadi **tidak diubah** menjadi objek `Post`.

RetrofitClient: Sekali untuk Semua

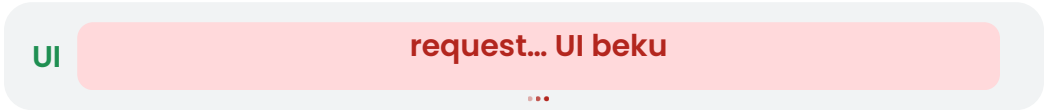
```
object RetrofitClient {
    private const val BASE_URL =
        "https://<kunci>.mockapi.io/api/v2/"

    private val loggingInterceptor =
        HttpLoggingInterceptor().apply {
            level = if (BuildConfig.DEBUG)
                Level.BODY else Level.NONE
        }
    private val okHttpClient = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor).build()

    private val retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .client(okHttpClient)
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    val apiService: ApiService =
        retrofit.create(ApiService::class.java)
}
```

- `object` = **singleton**, dipakai semua activity
- `baseUrl` **wajib diakhiri "/"** — kesalahan klasik
- `GsonConverterFactory` = jembatan JSON ↔ Kotlin
- `HttpLoggingInterceptor` cetak request/ response ke Logcat

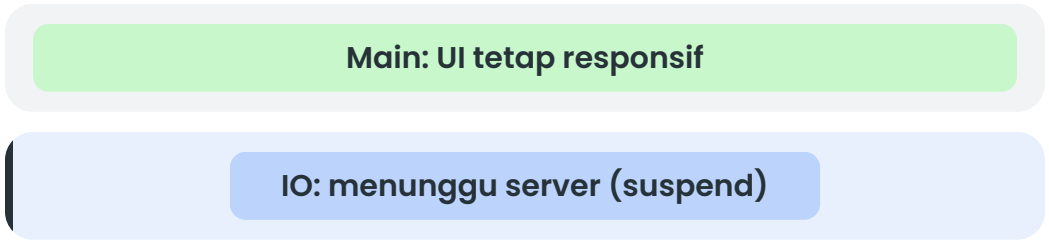
Dependency: retrofit, converter-gson, logging-interceptor, coroutines-android, lifecycle-runtime-ktx + izin `INTERNET`

Coroutines: Jaringan Tanpa Membeku

❌ Request di main thread



✅ Dengan coroutine



Tanpa `lifecycleScope.launch`, fungsi tidak berjalan.

```
lifecycleScope.launch {
    try {
        val posts = RetrofitClient
            .apiService.getPosts()
        postAdapter.updateData(posts)
    } catch (e: IOException) {
        // tidak ada internet
    } catch (e: HttpException) {
        // 4xx / 5xx
    }
}
```

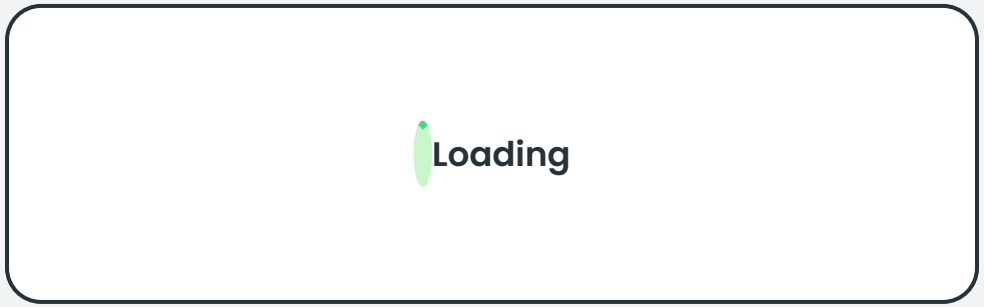
- `suspend` boleh “dijeda” tanpa memblokir UI
- Terikat siklus hidup Activity → bebas memory leak
- Kode jaringan ditulis **berurutan**, tanpa callback hell

Menampilkan Data & Navigasi

ListView + BaseAdapter

`getCount` , `getItem` , `getItemId` , `getView` .
`convertView` + tag = memakai kembali tampilan baris yang sudah ada saat daftar digulir, agar hemat memori. `notifyDataSetChanged()` = memberi tahu ListView bahwa data berubah, supaya tampilannya diperbarui.

Tiga status layar



Intent + Extra

`Intent(this, PostDetailActivity::class.java)`
`putExtra(EXTRA_POST_ID, id)` → `getStringExtra()` .
Muat ulang di `onResume()` .

ViewBinding

`viewBinding = true` → `ActivityMainBinding` dsb.
Akses view tanpa `findViewById` , dicek compiler.

Studi Kasus: Proyek Pertemuan 11

📋 List

GET semua post → ListView; tombol ke form POST

🔍 Detail

GET satu post; tombol PUT, PATCH (dialog), DELETE

✏️ Form

Satu activity dua mode: CREATE (POST) & EDIT (PUT)

0	1	2	3	4	5–6
Konfigurasi Gradle, izin INTERNET	**Lapisan data** Post, ApiService, RetrofitClient	**GET list** Adapter, MainActivity ✔ list muncul	**GET detail** Intent ✔ klik item	**POST** PostRequest, form ✔ buat post	**PUT/PATCH/ DELETE** ✔ CRUD penuh

Konfirmasi tiap aksi: dialog berisi status code + JSON respons (Gson pretty-print).

Rencana Kegiatan 3 Jam

Teori 20′	JSON 15′	Demo Bruno 25′	Praktik T0–2 45′	☕	T3–4 30′	T5–6 25′	QA

00:00–00:20	Teori API, client–server, REST & HTTP
00:20–00:35	JSON + model Kotlin
00:35–01:00	**Demo Bruno** ke MockAPI (GET/POST/PUT/PATCH/DELETE)
01:00–01:45	**Praktik Tahap 0–2**: konfigurasi, lapisan data, GET list
01:55–02:25	**Praktik Tahap 3–4**: detail + form POST (setelah istirahat 10′)
02:25–02:50	**Praktik Tahap 5–6**: PUT, PATCH, DELETE + uji CRUD penuh
02:50–03:00	Debugging umum, QA, penutup + tugas

Miskonsepsi & Jebakan Umum

“PATCH = PUT”

Demo Bruno membuktikan bedanya di JSON.

baseUrl tanpa “/”

Crash saat build.

Lupa izin INTERNET

Muncul `UnknownHostException`.

`id` dikira angka

Di MockAPI bertipe `String`.

`suspend` = ajaib

Tanpa `lifecycleScope.launch` tidak berjalan.

Nama field beda

Data jadi null / diabaikan diam-diam.

Glosarium & Referensi

- **API** pintu komunikasi antar program
- **Endpoint** URL resource
- **REST** arsitektur berbasis resource + HTTP method
- **JSON** format teks pertukaran data
- **Serialisasi / Deserialisasi** objek→JSON / JSON→objek
- **Retrofit** HTTP client type-safe
- **Gson** konversi JSON ↔ Kotlin
- **Coroutine** asynchronous Kotlin
- **Singleton** satu instance (`object`)
- **Mock server** server tiruan

Referensi

square.github.io/retrofit

github.com/google/gson

mockapi.io · usebruno.com

developer.mozilla.org → HTTP/Status

jsonplaceholder.typicode.com

developer.android.com/kotlin/coroutines

Terima kasih 🙌

Ada pertanyaan?