ENCS5150 · EXP 8 COMPANION

# Talking to a REST API

What an HTTP request and response contain, how JSON becomes Java objects, why the network never runs on the main thread, and what real apps use instead.

# Five questions

01 What is a REST API?

02 What is in a request, and in its response?

03 How does JSON become Java objects?

04 Why can’t the main thread wait for the network?

05 What do real apps use instead?

01 · REST

# Resources at addresses, verbs from HTTP

| Method | Means | Request | Typical answer |
|---|---|---|---|
| `GET` | Read | `GET /students` | 200 and the list |
| `POST` | Create | `POST /students + body` | 201 and the new student |
| `PUT` | Replace | `PUT /students/3 + body` | 200 and the student |
| `DELETE` | Delete | `DELETE /students/3` | 204, no body |

The address says **which** data, the method says **what to do** with it. Any client that speaks HTTP can use the same API: an app, a website, Postman.

02 · REQUEST AND RESPONSE

# What the app’s POST sends, and what comes back

## REQUEST

```
POST /users HTTP/1.1
Host: jsonplaceholder.typicode.com
Content-Type: application/json
Accept: application/json

{"name":"Ahmad","age":21}
```

Method and path · headers · a blank line · the body

## RESPONSE

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "name": "Ahmad",
  "age": 21,
  "id": 11
}
```

Status code · headers · a blank line · the body

02 · REQUEST AND RESPONSE

# The first digit of the status code tells you who to blame

2xx · SUCCESS

### It worked

`200 OK`: the answer is in the body.
`201 Created`: something new exists.

4xx · THE REQUEST

### Fix the app

`400`: the body is wrong.
`404`: nothing at that address.

5xx · THE SERVER

### Fix the server

`500`: it crashed handling the request.
`503`: it is down or overloaded.

Check the code before parsing the body: an error page is not a list of students.

# Arrays become lists, objects become Java objects

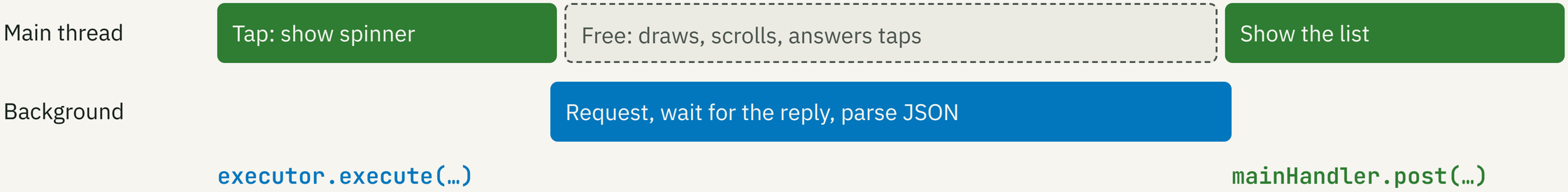
```
[
  { "id": 1,
    "name": "Ahmad",
    "age": 21 },
  { … }
]
```

| | |
|---|---|
| `[ … ]` | JSONArray: `length()`, `getJSONObject(i)` |
| `{ … }` | JSONObject: `getInt("id")`, `getString("name")` |
| **a field** | a field of `Student`, with the same name |

```
JSONObject o = array.getJSONObject(i);
new Student(o.getInt("id"), o.getString("name"),
            o.getInt("age"));
```

04 · THREADS

# Wait in the background, show the result on the main thread




| Break the rule | Android throws |
|---|---|
| Network on the main thread | `NetworkOnMainThreadException` |
| Touch a view from the background | `CalledFromWrongThreadException` |

04 · FAILURES

# Every request can fail. Say how.

| What happened | What Java gives you | What the user should see |
|---|---|---|
| No network | `UnknownHostException` | No internet connection |
| Server too slow | `SocketTimeoutException` | The server took too long |
| Wrong address | Code `404` | The server answered 404 |
| Unexpected reply | `JSONException` | Not the JSON we expected |

Set timeouts, check the code, catch the exceptions, and hide the spinner on **every** path. The manual did none of these: any failure crashed its app.

04 · PERMISSIONS

# Two lines of configuration you will need

EVERY NETWORK APP

## The INTERNET permission

```
<uses-permission android:name=
  "android.permission.INTERNET" />
```

Granted at install, no prompt. Missing it: `SecurityException`.

PLAIN HTTP, EXP 9

## Cleartext for one address

```
<domain-config
  cleartextTrafficPermitted="true">
  <domain>10.0.2.2</domain>
```

Since Android 9, only HTTPS by default. 10.0.2.2 is your laptop, seen from the emulator.

05 · IS THERE SOMETHING BETTER?

# Retrofit: describe the API, the library does the rest

```
public interface StudentApi {
    @GET("api/students.json")
    Call<List<Student>> getStudents();

    @POST("users")
    Call<Student> addStudent(@Body Student s);
}

// runs in the background, calls back on the main thread
api.getStudents().enqueue(callback);
```

| In this lab | Retrofit |
|---|---|
| HttpManager | Generated from the interface |
| StudentJsonParser | A converter (Gson, Moshi) |
| Executor + Handler | `enqueue` |

Writing it by hand once shows what the library does for you.

TAKEAWAYS

# Four things to remember

01 A request is a method, an address, headers and maybe a body; check the code.

02 JSON arrays become lists, objects become Java objects, field by field.

03 Network in the background, views on the main thread, a message for every failure.

04 Real apps use Retrofit, but it does exactly what you wrote by hand.