

Mobile Applications

5. REST APIs in Android

Boni García

boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2025/2026

uc3m | Universidad **Carlos III** de Madrid



Table of contents

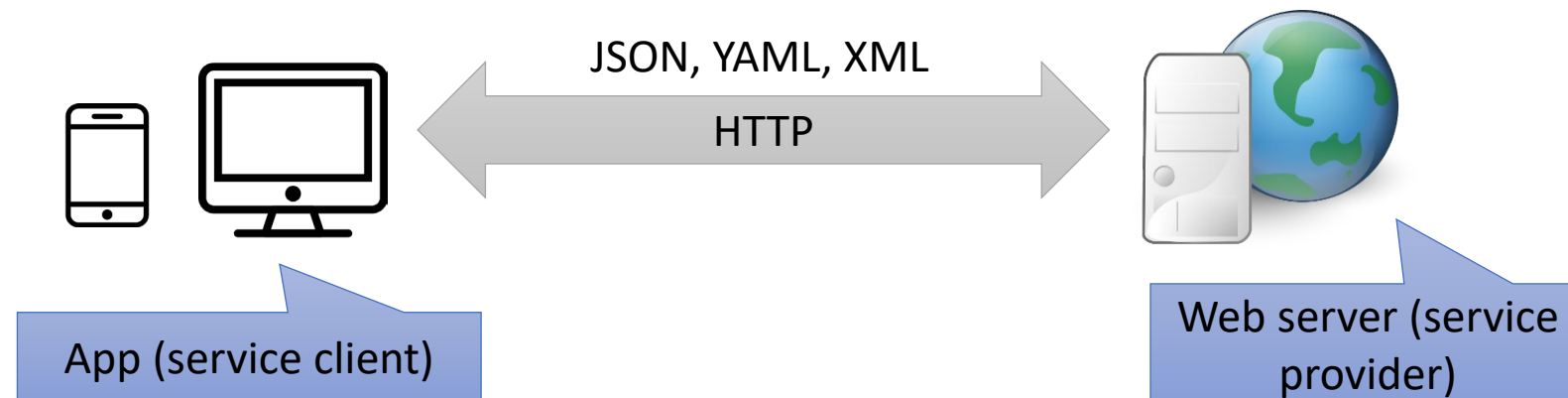
1. Introduction
2. HTTP fundamentals
3. REST APIs
4. REST clients in Android
5. Serverless REST API
6. Takeaways

1. Introduction

- Web applications:
 - Provides a service to end users:



- **Web services (also called **web APIs**):**
 - Provides a service to other programs:



1. Introduction

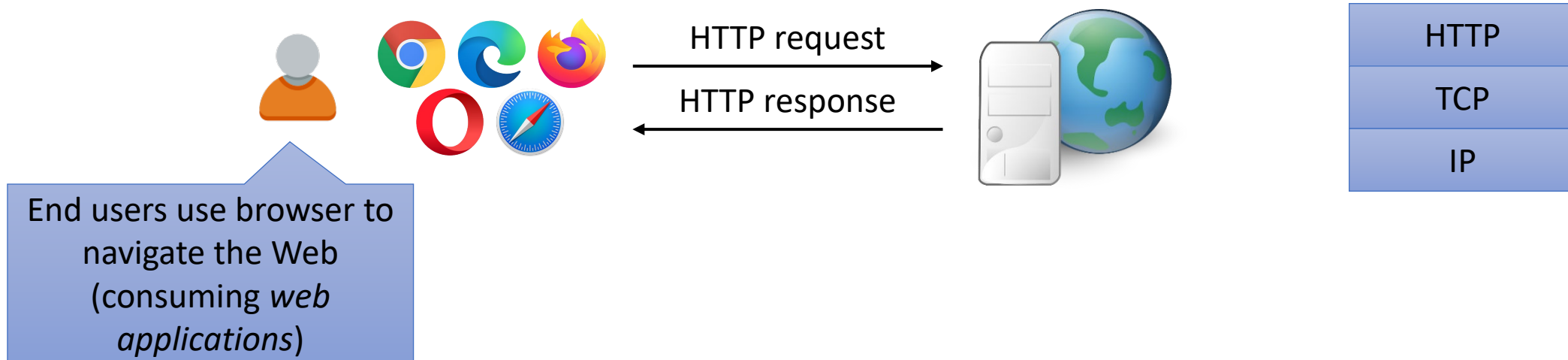
- A **web service** is a distributed service that allows applications to communicate over a network using HTTP
- Instead of returning web pages (HTML), web services return data in machine-readable formats such as:
 - JSON (most common)
 - XML
 - YAML
- Web services expose an API (Application Programming Interface) that allows other applications (e.g., web or mobile apps) to access and manipulate data
- There are different types of web services, such as SOAP and REST
- In this course, we focus on **REST**, since it is widely used today

Table of contents

1. Introduction
2. HTTP fundamentals
3. REST APIs
4. REST clients in Android
5. Serverless REST API
6. Takeaways

2. HTTP fundamentals

- HTTP (Hypertext Transfer Protocol) is an application layer protocol in the Internet reference model for transmitting hypermedia documents (i.e., web pages)
- HTTP is based on a client-server architecture:



2. HTTP fundamentals

- HTTP messages can be:
 - **Request** (from clients)
 - **Responses** (from servers)
- HTTP defines **methods** (sometimes referred to as *verbs*) to indicate the desired action to be performed on the identified resource
 - Common methods: GET (for reading), POST (for sending data)
- HTTP **headers** are a list of strings sent and received in request and response
 - Headers include extra information about the communication
- HTTP response **status codes** indicate whether a specific HTTP request has been successfully completed
 - Common examples: 200 ok, 404 not found, 500 internal server error

2. HTTP fundamentals

- Example of request-response:

HTTP request

```
Request line { GET /index.html HTTP/1.1
Headers      { Host: www.example.com
              { User-Agent: Mozilla/4.0
              { Accept: text/html, image/gif, image/jpeg
Empty line   {
```

HTTP response

```
Response line { HTTP/1.1 200 OK
Headers       { Date: Tue, 31 Dec 2023 23:59:59 GMT
              { Server: Apache/2.0.54 (Fedora)
              { Content-Type: text/html
              { Last-Modified: Mon, 30 Dec 2023 ...
              { Content-Length: 1221
Empty line    {
Body          { <html>
              {     <body>
              {     <h1>Example page</h1>
              {     . . .
              {     </body>
              { </html>
```

2. HTTP fundamentals

- **HTTPS** (Hypertext Transfer Protocol Secure) is the secure version of HTTP
 - With HTTPS it is achieved that sensitive information (passwords, etc.) cannot be intercepted by an attacker, since the only thing he will obtain will be an encrypted data stream that will be impossible for him/her to decrypt
- **TLS** (Transport Layer Security) is a protocol that provides encryption over TCP connections

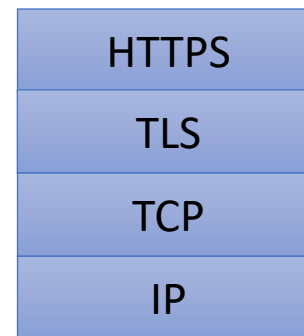


Table of contents

1. Introduction
2. HTTP fundamentals
- 3. REST APIs**
 - JSON
 - Tools
4. REST clients in Android
5. Serverless REST API
6. Takeaways

3. REST APIs

- **REST** (REpresentational State Transfer) is an architectural style for designing web services
 - REST is the most popular approach for building web APIs nowadays
- REST is built on top of HTTP and follows a client-server architecture
 - The server manages a set of **resources** and receives requests from clients
 - Each resource is identified by a unique URL, commonly called an **endpoint**
 - Each resource has a **representation**, that is, a machine-readable description of its current state
 - Resource representations are usually expressed using data-interchange formats such as **JSON**, YAML, or XML

3. REST APIs

- We use the **HTTP methods** (also called *verbs*) to map REST actions:

Most important methods (to implement CRUD operations)

HTTP Method	Description
GET	Read a resource
POST	Send a new resource to the server
PUT	Update a resource
DELETE	Eliminate a resource
PATCH	Update partially a resource
HEAD	Ask if a given resource exists without returning any of its representations
OPTIONS	Retrieve the available verbs for a given resource

3. REST APIs

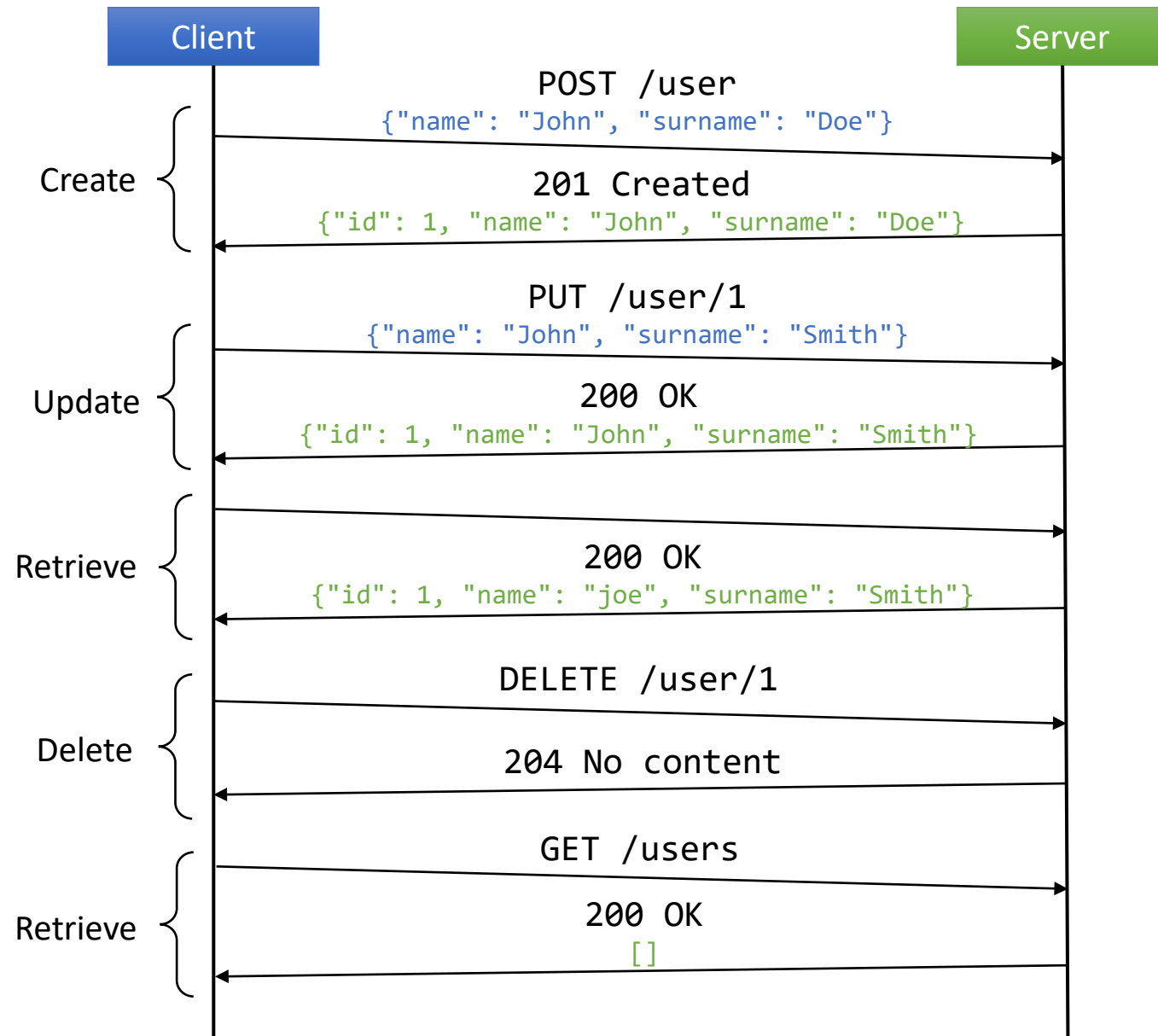
- We use the HTTP status codes to identify the response associated with REST actions

The following table summarizes the typical HTTP status code reused in REST

Status Code	Description
200 OK	The request was successful, and the content requested was returned (e.g., in a GET request)
201 Created	The resource was created (e.g., in a POST request)
204 No content	The action was successful, but no content was returned. This status code is useful in actions that do not require a response body (e.g., in a DELETE request)
301 Moved permanently	The resource was moved to another location
400 Bad request	The request has some problems (e.g., missing parameters)
401 Unauthorized	The requested resource is not accessible for the user that made the request
403 Forbidden	The resource is not accessible, but unlike 401, authentication will not affect the response
404 Not found	The provided endpoint does not identify any resource
405 Method not allowed	The used verb is not allowed (e.g., when using PUT in a read-only resource)
500 Internal server error	Generic unexpected condition in the server-side

3. REST APIs

The following example shows how REST API models operations as actions on resources using HTTP methods



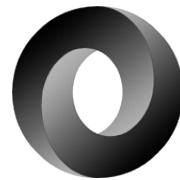
3. REST APIs - JSON

- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format
 - It is the most popular data formats for web APIs

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Trevor"
  ],
  "spouse": null
}
```

3. REST APIs - JSON

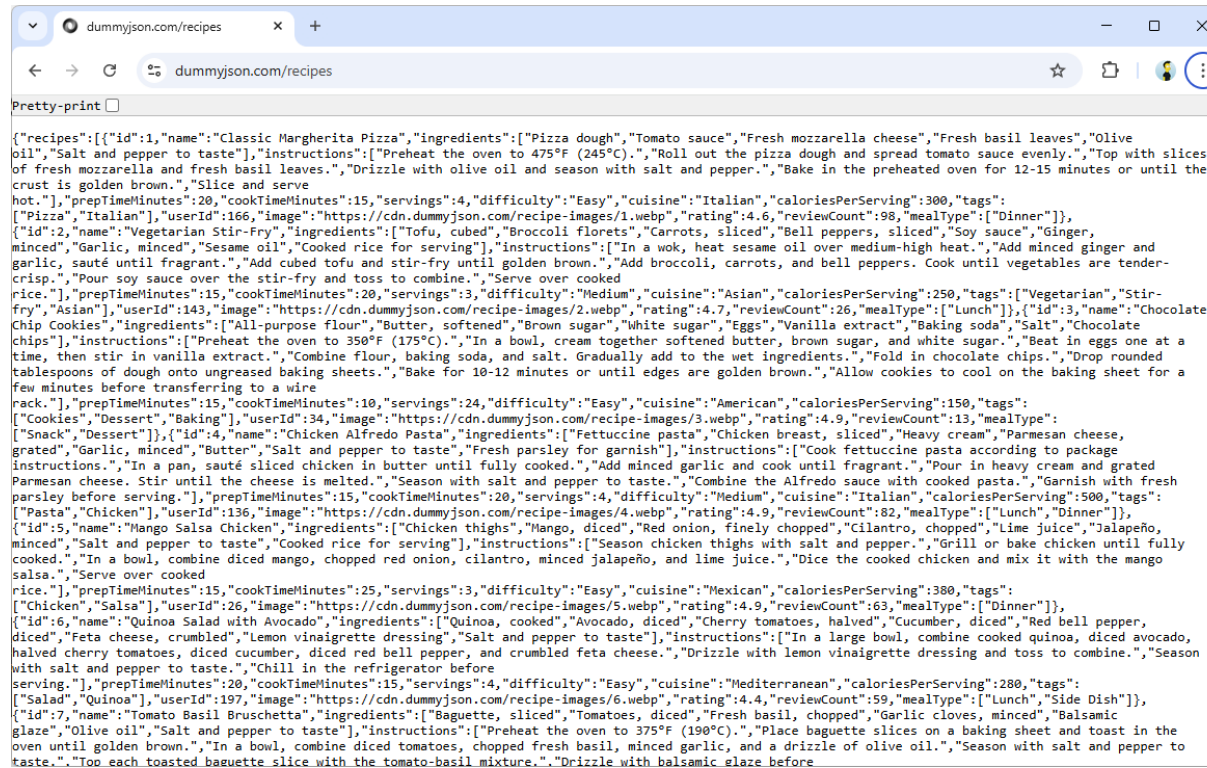
- JSON is defined as an open standard:
 - A collection of name/value pairs of objects
 - An object begins with the symbol { and ends with }
 - The name and value are separated by a colon (:)
 - An ordered list of values:
 - A list begins with the symbol [and ends with]
 - A comma (,) is used to separate the elements in a list
 - Values can be strings enclosed in double quotes (" "), numbers, boolean values (**true** or **false**), or **null**



<https://www.json.org/>

3. REST APIs - Tools

- We can use different tools to interact with REST services
- To making GET requests, we can use directly a web browser
 - For instance: <https://dummyjson.com/recipes>



```
pretty-print
{"recipes":[{"id":1,"name":"Classic Margherita Pizza","ingredients":["Pizza dough","Tomato sauce","Fresh mozzarella cheese","Fresh basil leaves","Olive oil","Salt and pepper to taste"],"instructions":["Preheat the oven to 475°F (245°C).","Roll out the pizza dough and spread tomato sauce evenly.","Top with slices of fresh mozzarella and fresh basil leaves.","Drizzle with olive oil and season with salt and pepper.","Bake in the preheated oven for 12-15 minutes or until the crust is golden brown."],"slice and serve hot."],"prepTimeMinutes":20,"cookTimeMinutes":15,"servings":4,"difficulty":"Easy","cuisine":"Italian","caloriesPerServing":300,"tags":["Pizza","Italian"],"userId":166,"image":"https://cdn.dummyjson.com/recipe-images/1.webp","rating":4.6,"reviewCount":98,"mealType":["Dinner"]}, {"id":2,"name":"Vegetarian Stir-Fry","ingredients":["Tofu, cubed","Broccoli florets","Carrots, sliced","Bell peppers, sliced","Soy sauce","Ginger, minced","Garlic, minced","Sesame oil","Cooked rice for serving"],"instructions":["In a wok, heat sesame oil over medium-high heat."],"add minced ginger and garlic, sauté until fragrant."],"add cubed tofu and stir-fry until golden brown."],"add broccoli, carrots, and bell peppers. Cook until vegetables are tender-crisp."],"pour soy sauce over the stir-fry and toss to combine."],"serve over cooked rice."],"prepTimeMinutes":15,"cookTimeMinutes":20,"servings":3,"difficulty":"Medium","cuisine":"Asian","caloriesPerServing":250,"tags":["Vegetarian","Stir-fry","Asian"],"userId":143,"image":"https://cdn.dummyjson.com/recipe-images/2.webp","rating":4.7,"reviewCount":26,"mealType":["Lunch"]}, {"id":3,"name":"Chocolate Chip Cookies","ingredients":["All-purpose flour","Butter, softened","Brown sugar","White sugar","Eggs","Vanilla extract","Baking soda","Salt","Chocolate chips"],"instructions":["Preheat the oven to 350°F (175°C).","In a bowl, cream together softened butter, brown sugar, and white sugar."],"beat in eggs one at a time, then stir in vanilla extract."],"combine flour, baking soda, and salt. Gradually add to the wet ingredients."],"fold in chocolate chips."],"drop rounded tablespoons of dough onto ungreased baking sheets."],"bake for 10-12 minutes or until edges are golden brown."],"allow cookies to cool on the baking sheet for a few minutes before transferring to a wire rack."],"prepTimeMinutes":15,"cookTimeMinutes":10,"servings":24,"difficulty":"Easy","cuisine":"American","caloriesPerServing":150,"tags":["Cookies","Dessert","Baking"],"userId":34,"image":"https://cdn.dummyjson.com/recipe-images/3.webp","rating":4.9,"reviewCount":13,"mealType":["Snack","Dessert"]}, {"id":4,"name":"Chicken Alfredo Pasta","ingredients":["Fettuccine pasta","Chicken breast, sliced","Heavy cream","Parmesan cheese, grated","Garlic, minced","Butter","Salt and pepper to taste","Fresh parsley for garnish"],"instructions":["Cook fettuccine pasta according to package instructions."],"in a pan, sauté sliced chicken in butter until fully cooked."],"add minced garlic and cook until fragrant."],"pour in heavy cream and grated Parmesan cheese. Stir until the cheese is melted."],"season with salt and pepper to taste."],"combine the Alfredo sauce with cooked pasta."],"garnish with fresh parsley before serving."],"prepTimeMinutes":15,"cookTimeMinutes":20,"servings":4,"difficulty":"Medium","cuisine":"Italian","caloriesPerServing":500,"tags":["Pasta","Chicken"],"userId":136,"image":"https://cdn.dummyjson.com/recipe-images/4.webp","rating":4.9,"reviewCount":82,"mealType":["Lunch","Dinner"]}, {"id":5,"name":"Mango Salsa Chicken","ingredients":["Chicken thighs","Mango, diced","Red onion, finely chopped","Cilantro, chopped","Lime juice","Jalapeño, minced","Salt and pepper to taste","Cooked rice for serving"],"instructions":["Season chicken thighs with salt and pepper."],"grill or bake chicken until fully cooked."],"in a bowl, combine diced mango, chopped red onion, cilantro, minced jalapeño, and lime juice."],"dice the cooked chicken and mix it with the mango salsa."],"serve over cooked rice."],"prepTimeMinutes":15,"cookTimeMinutes":25,"servings":3,"difficulty":"Easy","cuisine":"Mexican","caloriesPerServing":380,"tags":["Chicken","Salsa"],"userId":26,"image":"https://cdn.dummyjson.com/recipe-images/5.webp","rating":4.9,"reviewCount":63,"mealType":["Dinner"]}, {"id":6,"name":"Quinoa Salad with Avocado","ingredients":["Quinoa, cooked","Avocado, diced","Cherry tomatoes, halved","Cucumber, diced","Red bell pepper, diced","Feta cheese, crumbled","Lemon vinaigrette dressing","Salt and pepper to taste"],"instructions":["In a large bowl, combine cooked quinoa, diced avocado, halved cherry tomatoes, diced cucumber, diced red bell pepper, and crumbled feta cheese."],"drizzle with lemon vinaigrette dressing and toss to combine."],"season with salt and pepper to taste."],"chill in the refrigerator before serving."],"prepTimeMinutes":20,"cookTimeMinutes":15,"servings":4,"difficulty":"Easy","cuisine":"Mediterranean","caloriesPerServing":280,"tags":["Salad","Quinoa"],"userId":197,"image":"https://cdn.dummyjson.com/recipe-images/6.webp","rating":4.4,"reviewCount":59,"mealType":["Lunch","Side Dish"]}, {"id":7,"name":"Tomato Basil Bruschetta","ingredients":["Baguette, sliced","Tomatoes, diced","Fresh basil, chopped","Garlic cloves, minced","Balsamic glaze","Olive oil","Salt and pepper to taste"],"instructions":["Preheat the oven to 375°F (190°C).","Place baguette slices on a baking sheet and toast in the oven until golden brown."],"in a bowl, combine diced tomatoes, chopped fresh basil, minced garlic, and a drizzle of olive oil."],"season with salt and pepper to taste."],"top each toasted baguette slice with the tomato-basil mixture."],"drizzle with balsamic glaze before"}]
```

[DummyJSON](https://dummyjson.com) is a public REST service for testing and prototyping

3. REST APIs - Tools

- For more complex REST operations (e.g. POST, with custom headers, body, etc.) we can use for instance:
 - Postman (desktop app): <https://www.postman.com/>

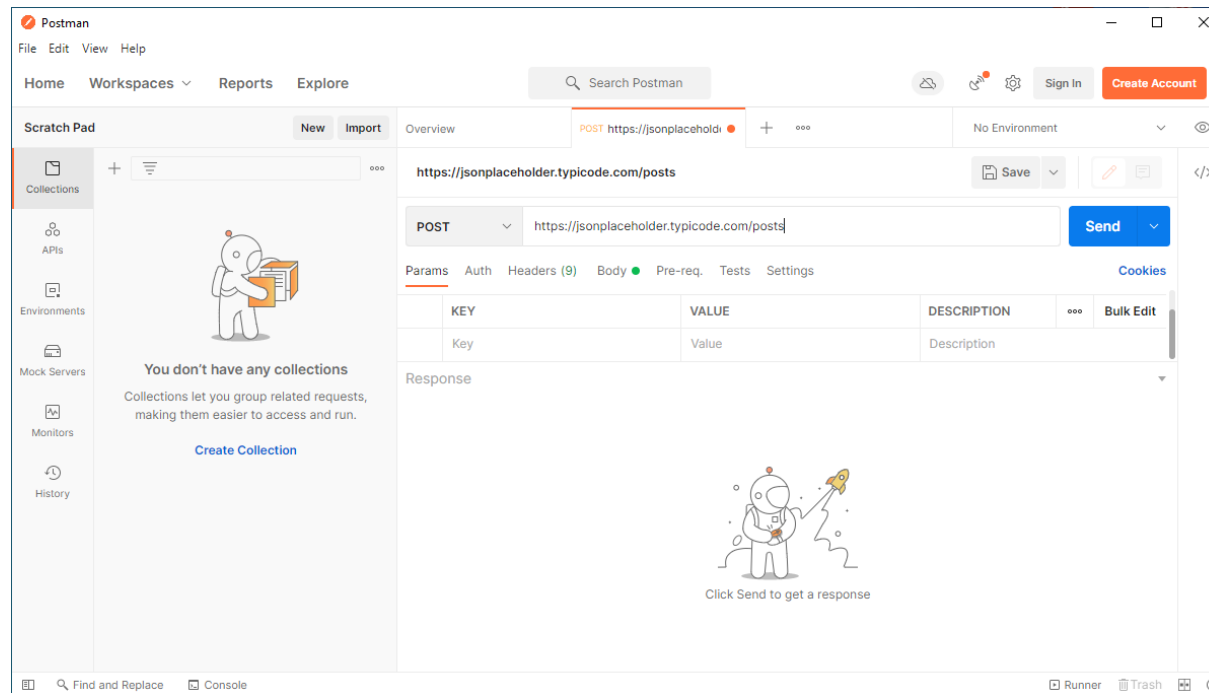


Table of contents

1. Introduction
2. HTTP fundamentals
3. REST APIs
- 4. REST clients in Android**
 - Retrofit
5. Serverless REST API
6. Takeaways

4. REST clients in Android

- Implementing a REST client in an Android app using Kotlin and Jetpack Compose involves several steps
 - Grant connectivity permissions to our app

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

To be included in the manifest

- Set up dependencies for networking and JSON parsing
- Define data models for the API (request/response)
- Use a REST client (e.g., Retrofit) to interact with the REST API
- Use a coroutine-based approach for asynchronous network calls (e.g., in a ViewModel)
- Display the data in a Compose UI

4. REST clients in Android - Retrofit

- There are different high-level specific libraries implementing REST clients in Java/Kotlin, such as:
 - Retrofit: <https://square.github.io/retrofit/>
 - Jersey: <https://eclipse-ee4j.github.io/jersey/>
 - RESTEasy: <https://resteasy.dev/>
 - OkHttp: <https://square.github.io/okhttp/>
- For the following examples, we use **Retrofit**:

build.gradle.kts (app)

```
implementation(Libs.retrofit)
implementation(Libs.converter.gson)
```

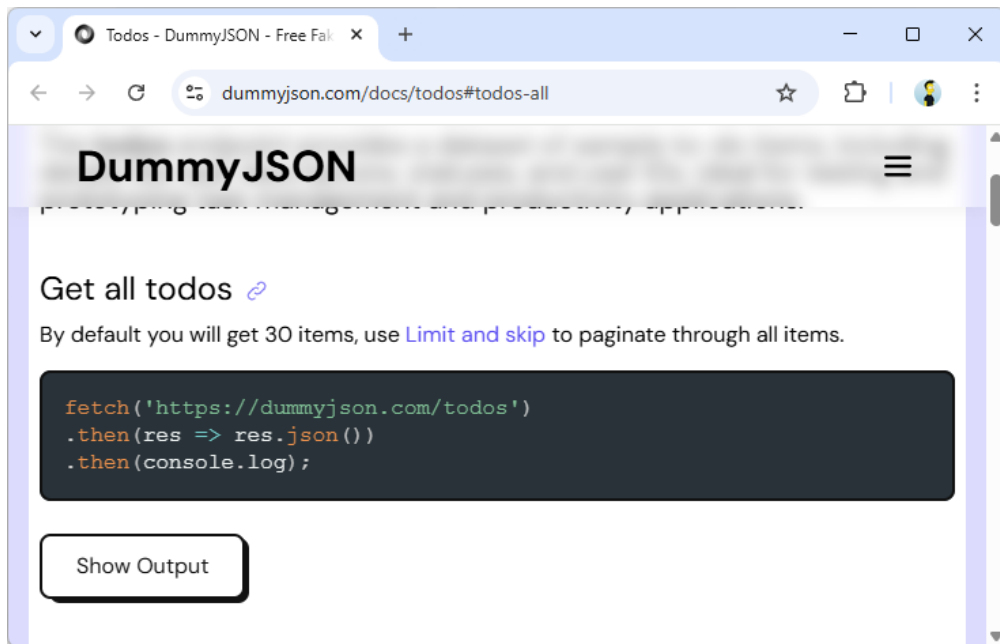
libs.version.toml

```
[versions]
retrofit = "3.0.0"
converterGson = "3.0.0"

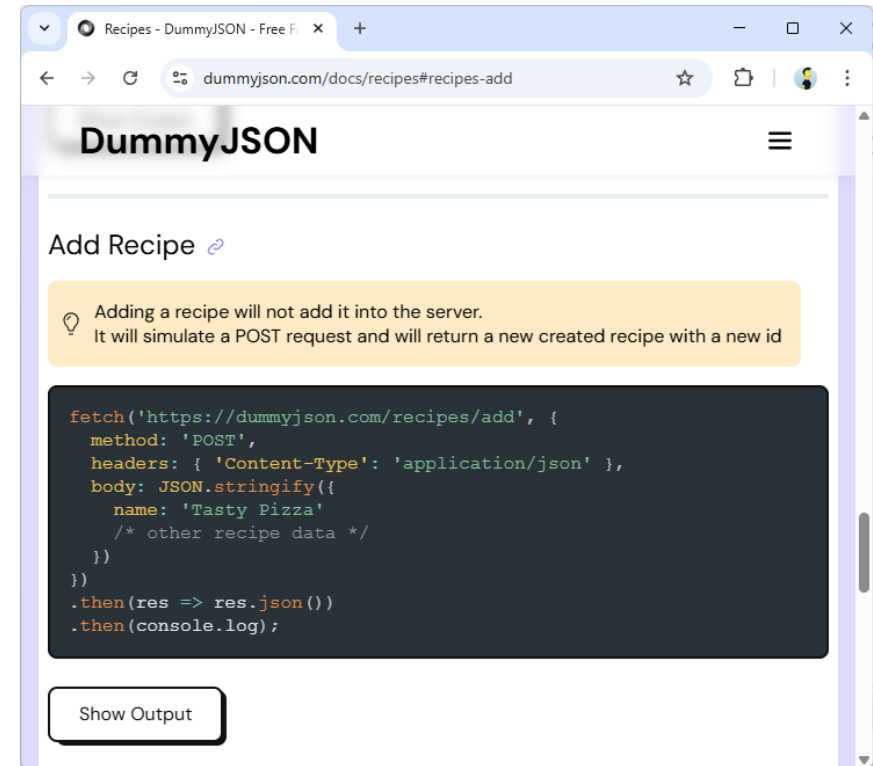
[libraries]
retrofit = { module = "com.squareup.retrofit2:retrofit", version.ref = "retrofit" }
converter-gson = { module = "com.squareup.retrofit2:converter-gson", version.ref = "converterGson" }
```

4. REST clients in Android - Retrofit

- The following sample app makes GET and POST requests to the REST service [DummyJSON](https://dummyjson.com/) using Retrofit



<https://dummyjson.com/docs/todos#todos-all>



<https://dummyjson.com/docs/recipes#recipes-add>

4. REST clients in Android - Retrofit

```
data class Todos(  
    val todos: List<Todo>,  
    val total: Long,  
    val skip: Long,  
    val limit: Long,  
)  
  
data class Todo(  
    val id: Long,  
    val todo: String,  
    val completed: Boolean,  
    val userId: Long,  
)  
  
data class Recipe(  
    val id: Long? = null,  
    val name: String,  
    val ingredients: String,  
)
```

```
interface DummyJsonService {  
  
    @GET("todos")  
    suspend fun getTodos(): Response<Todos>  
  
    @POST("recipes/add")  
    suspend fun addRecipe(@Body recipe: Recipe): Response<Recipe>  
  
}
```

We define an interface containing all the endpoints (using `suspend`) we want to use (without the base URL)

```
object DummyJsonClient {  
    private const val BASE_URL = "https://dummyjson.com/"  
  
    val apiService: DummyJsonService by lazy {  
        Retrofit.Builder()  
            .baseUrl(BASE_URL)  
            .addConverterFactory(GsonConverterFactory.create())  
            .build()  
            .create(DummyJsonService::class.java)  
    }  
  
}
```

We create an instance of the previous interface like this

Data model. It can be automatically generated from the JSON using an online tool like: <https://transform.tools/json-to-kotlin>

4. REST clients in Android - Retrofit

```
class RestViewModel : ViewModel() {  
  
    // ...  
  
    fun fetchTodos() {  
        viewModelScope.Launch {  
            _isLoading.value = true  
            try {  
                val response = DummyJsonClient.apiService.getTodos()  
                if (response.isSuccessful) {  
                    _todos.value = response.body()?.todos ?: emptyList()  
                }  
            } catch (e: Exception) {  
                _snackMessage.value = e.message  
            } finally {  
                _isLoading.value = false  
            }  
        }  
    }  
  
    fun addRecipe(recipe: Recipe) {  
        viewModelScope.Launch {  
            try {  
                val response = DummyJsonClient.apiService.addRecipe(recipe)  
                _snackMessage.value = response.code().toString() + " " + response.message()  
            } catch (e: Exception) {  
                _snackMessage.value = e.message  
            }  
        }  
    }  
}
```

We use the previous API service from a ViewModel

4. REST clients in Android - Retrofit

```
@Composable
fun mainScreen(viewModel: RestViewModel = viewModel()) {
    val todos by viewModel.todos.collectAsState()
    val isLoading by viewModel.isLoading.collectAsState()
    val snackbarMessage by viewModel.snackbarMessage.collectAsState()

    var showDialog by remember { mutableStateOf(false) }
    var isHome by remember { mutableStateOf(true) }
    var fetching by remember { mutableStateOf(false) }
    val snackbarHostState = remember { SnackbarHostState() }

    Scaffold(floatingActionButton = {
        FloatingActionButton(onClick = { showDialog = true }) {
            Icon(Icons.Default.Add, contentDescription = stringResource(R.string.add))
        }
    }, snackbarHost = { SnackbarHost(hostState = snackbarHostState) }) { padding ->
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(padding),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            if (isHome) {
                Button(onClick = {
                    isHome = false
                }) {
                    Text(stringResource(R.string.get_todos))
                }
            } else if (!fetching) {
                viewModel.fetchTodos()
                fetching = true
            } else if (isLoading) {
                CircularProgressIndicator()
            } else {
                // ...
            }
        }
    }
}
```

Finally, we observe the changes in the ViewModel and display the responses in the UI

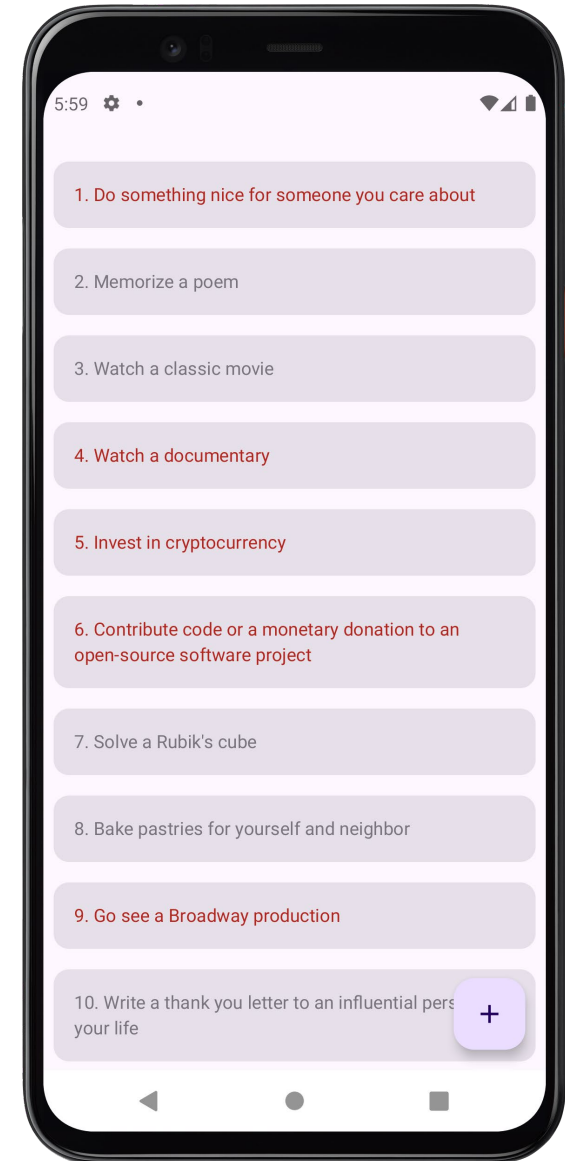


Table of contents

1. Introduction
2. HTTP fundamentals
3. REST APIs
4. REST clients in Android
- 5. Serverless REST API**
 - Node.js
 - Cloud functions
6. Takeaways

5. Serverless REST API

- **Serverless computing** is a model where developers run backend code without managing servers (the cloud provider handles the infrastructure)
- **Cloud Functions** is a serverless platform provided by Firebase that allows us to run backend code in response to HTTP requests and other events
 - Cloud Functions can be implemented using JavaScript, TypeScript, Python, or Dart
 - In this course, we use **JavaScript** to implement the server-side logic of a simple REST API

<https://firebase.google.com/docs/functions/>

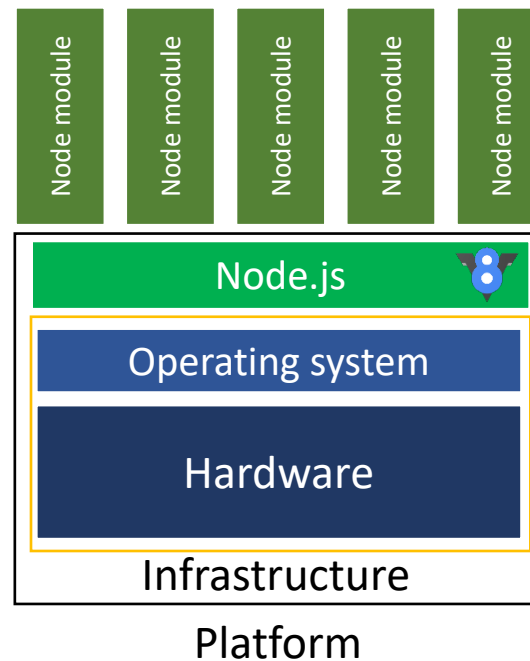
5. Serverless REST API - Node.js

- **Node.js** is an open-source, cross-platform JavaScript runtime environment that enables the execution of JavaScript code outside a web browser
 - Node.js runs on the V8 JavaScript engine, which is an open-source JavaScript engine developed by the Chromium Project



<https://nodejs.org/>

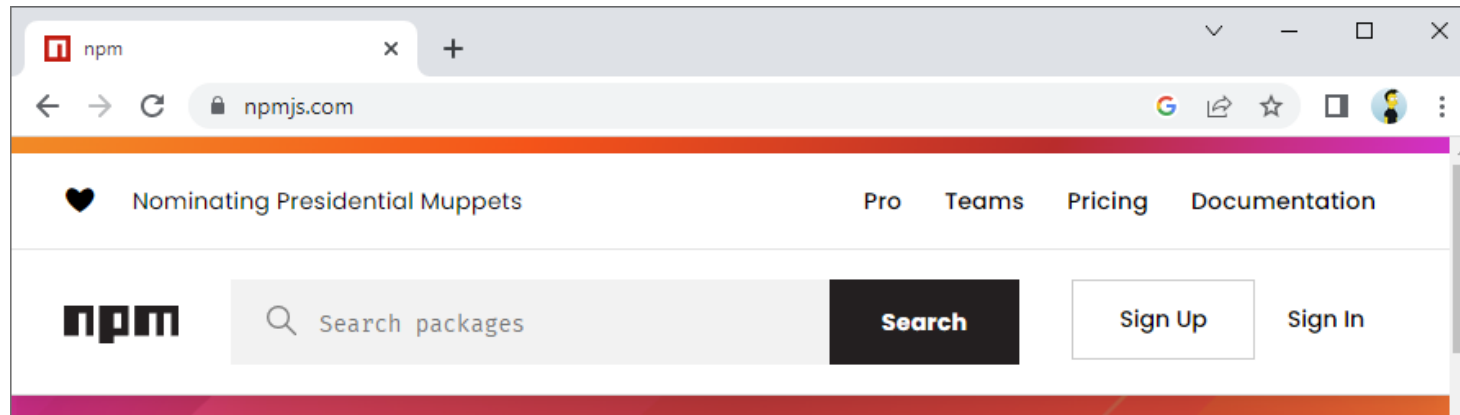
We can download the Node.js installer from its website



We can see Node.js as a software layer that allows to execute JavaScript apps

5. Serverless REST API - Node.js

- **NPM** is a package manager for Node.js packages (called Node modules)
 - It consists of a command line client (npm), and an online database of public and private packages, called the npm registry



- The NPM command line program is available after installing Node.js

```
> node --version
v24.14.0

> npm --version
11.11.1
```



<https://www.npmjs.com/>

5. Serverless REST API - Cloud functions

- In this course, we can create a cloud function as follows:
 1. Create a Firebase project
 - We already have Firebase projects in this course (**uc3m-it-2026-16504-g**-***)
 2. Set up Firebase CLI (Command Line Interface)
 - It is a command line tool provided by Google for interacting with Firebase services
 3. Login in Firebase
 4. Initialize project
 5. Implement the cloud function
 - For instance, using in JavaScript
 6. Emulate the cloud function locally
 7. Deploy to Firebase

<https://firebase.google.com/docs/functions/get-started>

5. Serverless REST API - Cloud functions

2. Set up Firebase CLI:

```
> npm install -g firebase-tools
changed 745 packages in 20s

> firebase --version
15.11.1
```

We use npm to install Firebase CLI

3. Login in Firebase:

```
> firebase login
i  Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our products. Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to identify you.
? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? No

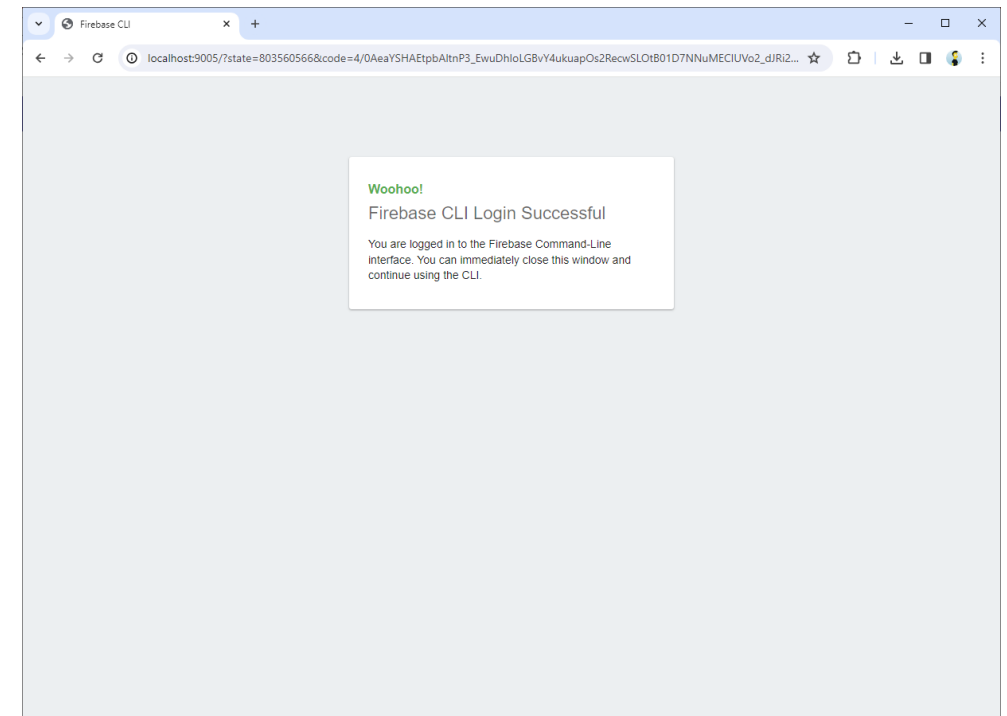
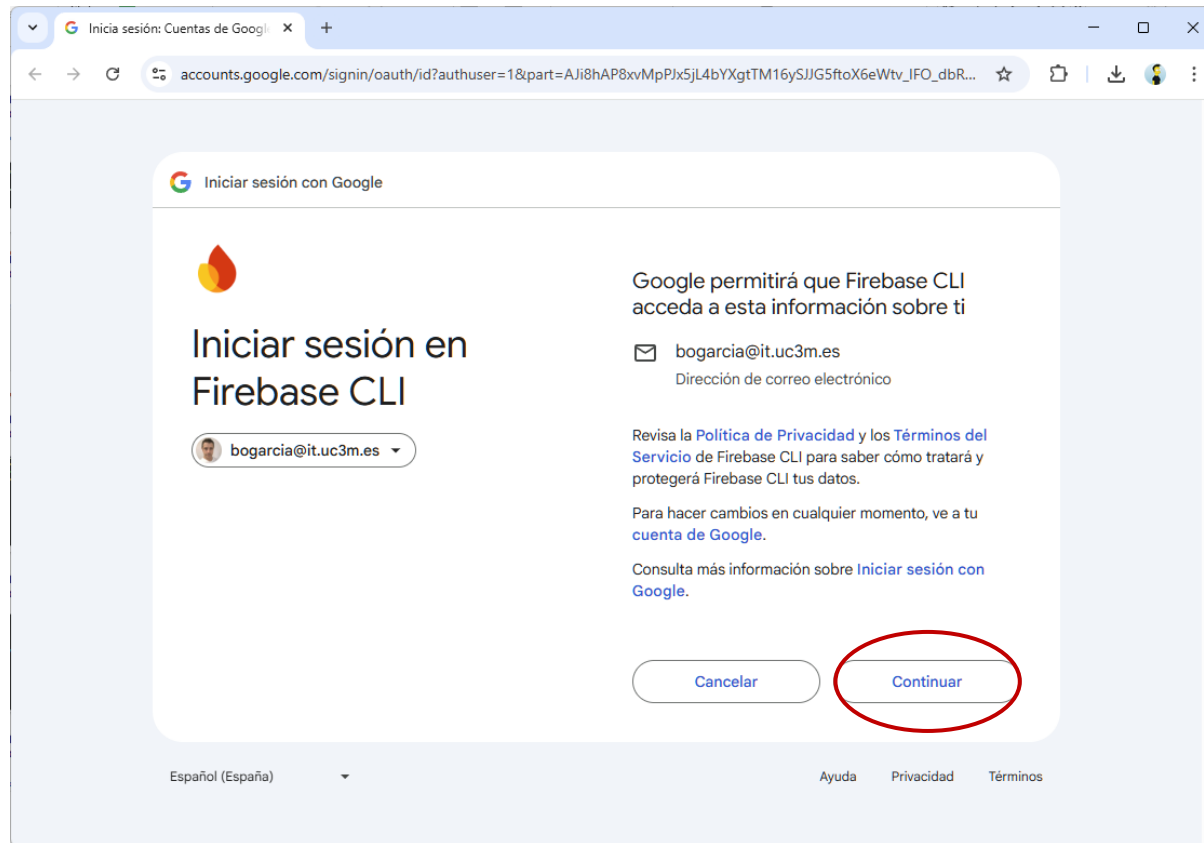
Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent.com&...

Waiting for authentication...

+ Success! Logged in as bogarcia@it.uc3m.es
```

5. Serverless REST API - Cloud functions

3. Login in Firebase:



5. Serverless REST API - Cloud functions

4. Initialize project:

```
C:\Users\boni\Documents\dev\cloud-functions-hello-world>firebase init functions
```

```
#####  #####  #####  #####  #####  #####  #####  #####  
##      ##  ##      ##  ##      ##      ##  ##  ##  ##      ##  
#####  ##  #####  #####  #####  #####  #####  #####  
##      ##  ##      ##  ##      ##      ##  ##  ##      ##  ##  
##      #####  ##      ##  #####  #####  ##      ##  #####  #####
```

You're about to initialize a Firebase project in this directory:

```
C:\Users\boni\Documents\dev\cloud-functions-hello-world
```

? Are you ready to proceed? **Yes**

=== Project Setup

First, let's associate this project directory with a Firebase project. You can create multiple project aliases by running `firebase use --add`, but for now we'll just set up a default project.

? Please select an option: **Use an existing project**

? Select a default Firebase project for this directory:

```
> uc3m-it-2026-13345-professors (uc3m-it-2026-16504-professors)  
uc3m-it-2025-16504-professors (uc3m-it-2024-16504-professors)
```

5. Serverless REST API - Cloud functions

4. Initialize project:

```
=== Functions Setup
Let's create a new codebase for your functions.
A directory corresponding to the codebase will be created in your project
with sample code pre-configured.
See https://firebase.google.com/docs/functions/organize-functions for
more information on organizing your functions using codebases.

Functions can be deployed with firebase deploy.

? What language would you like to use to write Cloud Functions? (Use arrow keys)
> JavaScript
  TypeScript
  Python
  Dart

? Do you want to use ESLint to catch probable bugs and enforce style? No
+ Wrote functions/package.json
+ Wrote functions/index.js
+ Wrote functions/.gitignore
? Do you want to install dependencies with npm now? Yes

added 530 packages, and audited 531 packages in 32s

i Writing configuration info to firebase.json...
i Writing project information to .firebaserc...
i Writing gitignore file to .gitignore...

+ Firebase initialization complete!
```

5. Serverless REST API - Cloud functions

5. Implement the cloud function:

- You can find a complete project example in GitHub:

<https://github.com/bonigarcia/cloud-functions-hello-world>

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");
const express = require("express");

admin.initializeApp();

const logger = functions.logger;
const app = express();
const db = admin.firestore();

// Hello world endpoint (GET)
app.get("/hello-world", (req, res) => {
  logger.log("Hello world received");
  return res.status(200).send("Hello world!");
});

// ...

exports.app = functions.https.onRequest(app);
```

5. Serverless REST API - Cloud functions

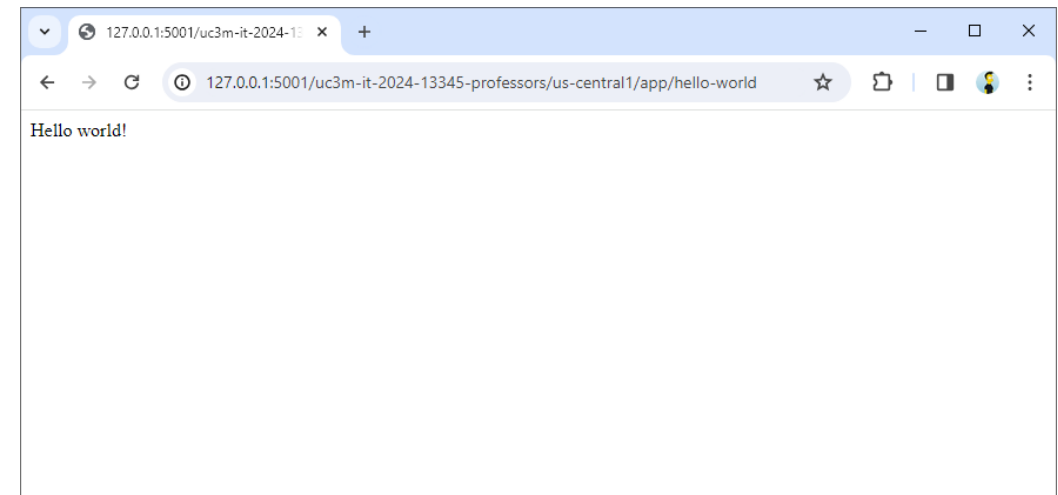
6. Emulate the cloud function locally:

```
> firebase emulators:start
i emulators: Starting emulators: functions, firestore
i firestore: Firestore Emulator logging to firestore-debug.log
+ firestore: Firestore Emulator UI websocket is running on 9150.
i ui: Emulator UI logging to ui-debug.log
i functions: Watching "C:\Users\boni\Documents\dev\cloud-functions-hello-world\functions" for Cloud Functions...
+ functions: Using node@20 from host.
Serving at port 8490

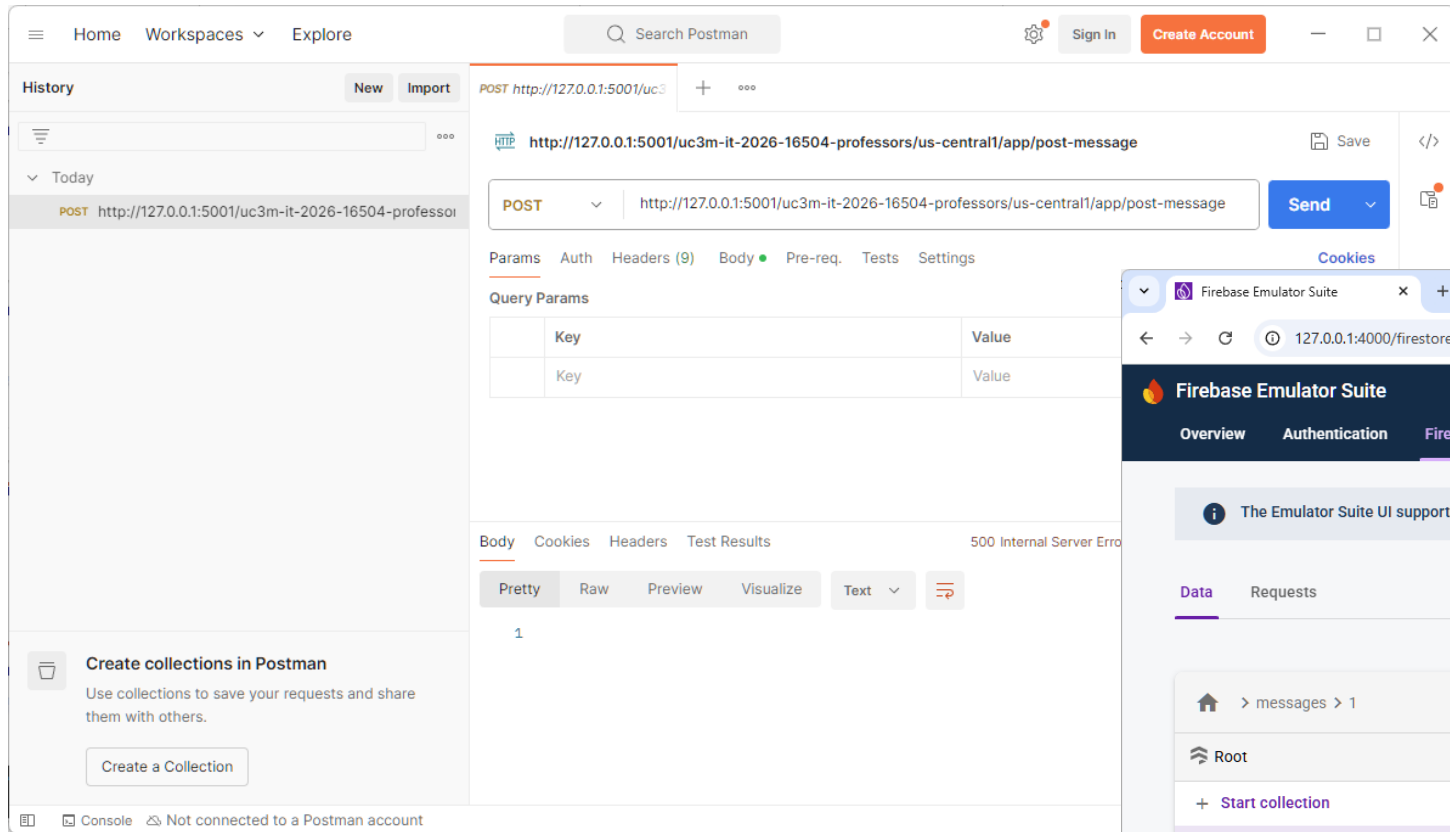
+ functions: Loaded functions definitions from source: app.
+ functions[us-central1-app]: http function initialized
(http://127.0.0.1:5001/uc3m-it-2025-13345-professors/us-central1/app).
```

✓ All emulators ready! It is now safe to connect your app.
i View Emulator UI at <http://127.0.0.1:4000/>

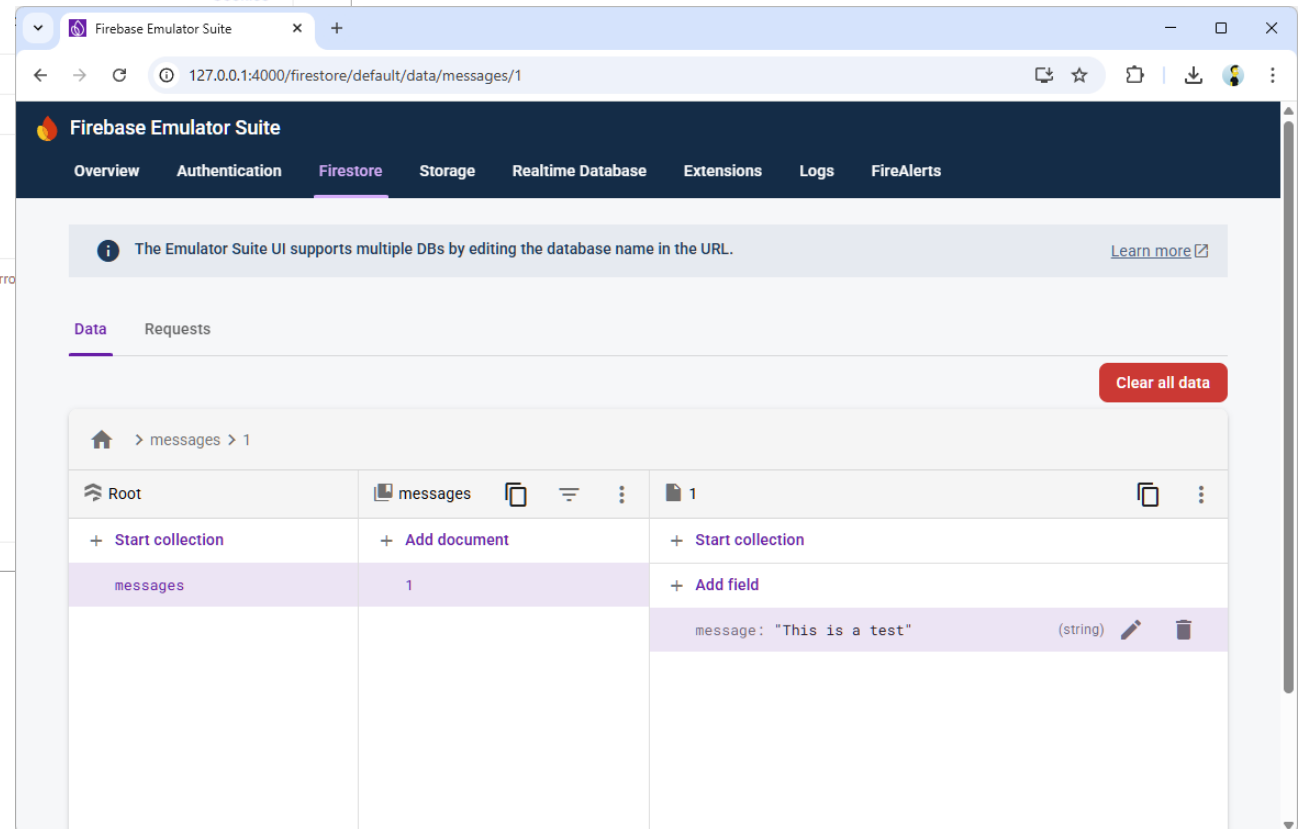
Emulator	Host:Port	View in Emulator UI
Functions	127.0.0.1:5001	http://127.0.0.1:4000/functions
Firestore	127.0.0.1:8080	http://127.0.0.1:4000/firestore



5. Serverless REST API - Cloud functions



Now we can interact with the locally deployed REST API



5. Serverless REST API - Cloud functions

7. Deploy to Firebase:

```
> firebase deploy --only functions
```

```
=== Deploying to 'uc3m-it-2026-16504-professors'...
```

```
i deploying functions
```

```
i functions: preparing codebase default for deployment
```

```
i functions: ensuring required API cloudfunctions.googleapis.com is
```

```
i functions: ensuring required API cloudbuild.googleapis.com is enab
```

```
i artifactregistry: ensuring required API artifactregistry.googleapi
```

```
+ artifactregistry: required API artifactregistry.googleapis.com is
```

```
+ functions: required API cloudbuild.googleapis.com is enabled
```

```
+ functions: required API cloudfunctions.googleapis.com is enabled
```

```
i functions: Loading and analyzing source code for codebase default
```

```
Serving at port 8497
```

```
i functions: preparing functions directory for uploading...
```

```
i functions: packaged C:\Users\boni\Documents\dev\cloud-functions-hello-world\functions (66.84 KB) for uploading
```

```
+ functions: functions folder uploaded successfully
```

```
i functions: creating Node.js 20 (1st Gen) function app(us-central1)...
```

```
+ functions[app(us-central1)] Successful create operation.
```

```
Function URL (app(us-central1)): https://us-central1-uc3m-it-2026-16504-professors.cloudfunctions.net/app
```

```
i functions: cleaning up build files...
```

```
+ Deploy complete!
```

```
Project Console: https://console.firebase.google.com/project/uc3m-it-2026-16504-professors/overview
```

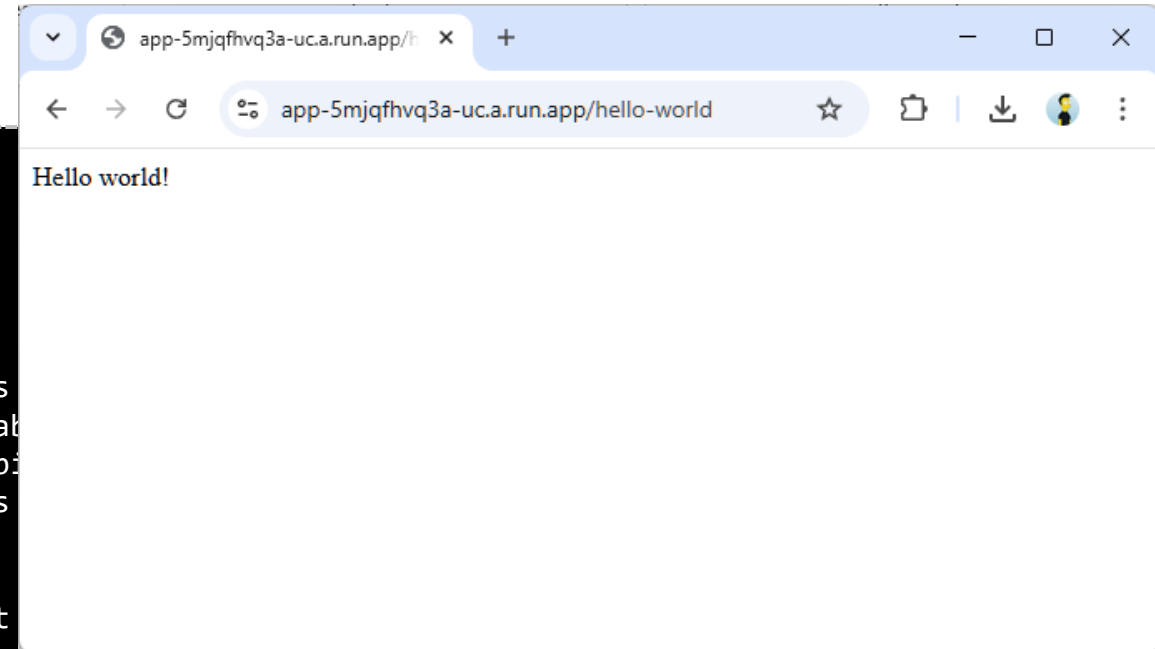


Table of contents

1. Introduction
2. HTTP
3. REST APIs
4. REST clients in Android
5. Serverless REST API
- 6. Takeaways**

6. Takeaways

- A **web API** is a type of distributed service allows applications to communicate over the network using **HTTP**
- **REST APIs** use HTTP (methods, response code, etc.) to access and manipulate resources (shared data)
- Resources in REST APIs are typically exchanged in **JSON** format
- In Android (client-side), we can consume REST APIs using libraries such as **Retrofit**
- In the server-side, we can implement REST APIs using **Cloud Functions** (serverless backend) from Firebase