Mobile Applications

5. Using web services in Android

Boni García

boni.garcia@uc3m.es

Telematic Engineering Department
School of Engineering

2024/2025

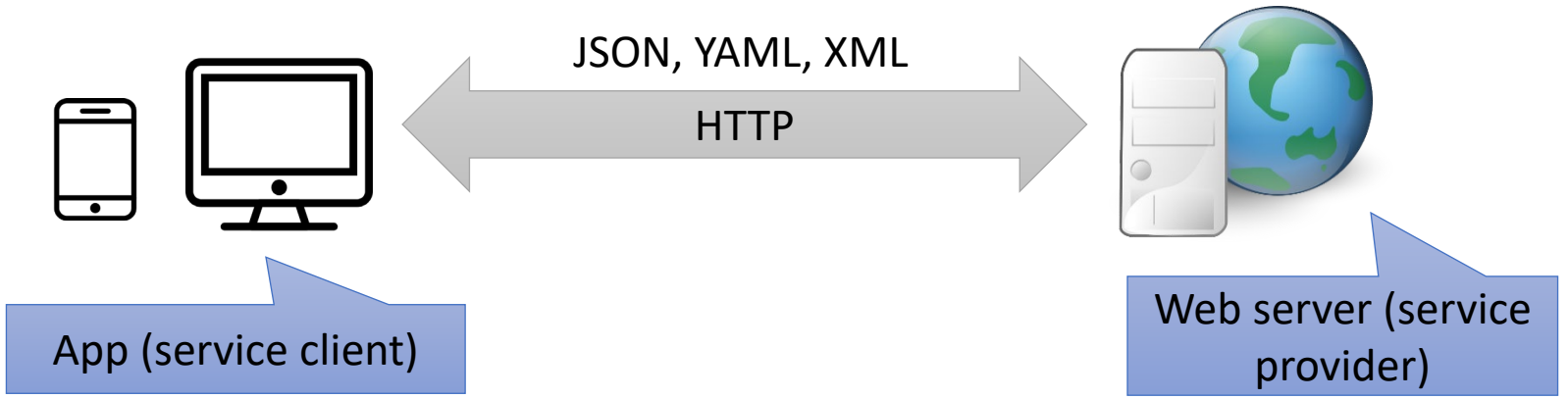
uc3m | Universidad Carlos III de Madrid

Table of contents

1. Introduction

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

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



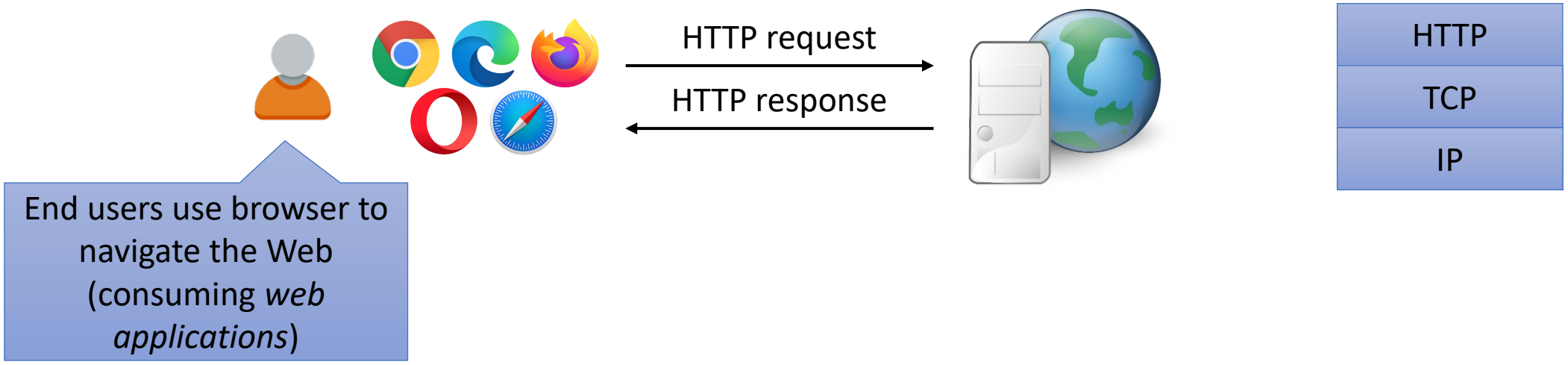
1. Introduction

- A **web service** is a distributed service that built on the top of HTTP
- Instead of sharing regular web resources (e.g., HTML documents), web services transfers data into machine-readable file formats such as JSON, YAML, or XML
- Web services provide an Application Programming Interface (API) for sharing resources (e.g. some data into a database) used for example by another by some software app (e.g., mobile app, a desktop app, or another server)
- There are different types of web services, such as SOAP or **REST**
  - We focus only in REST in this course, since it is widespread nowadays

Table of contents

2. HTTP

- 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:

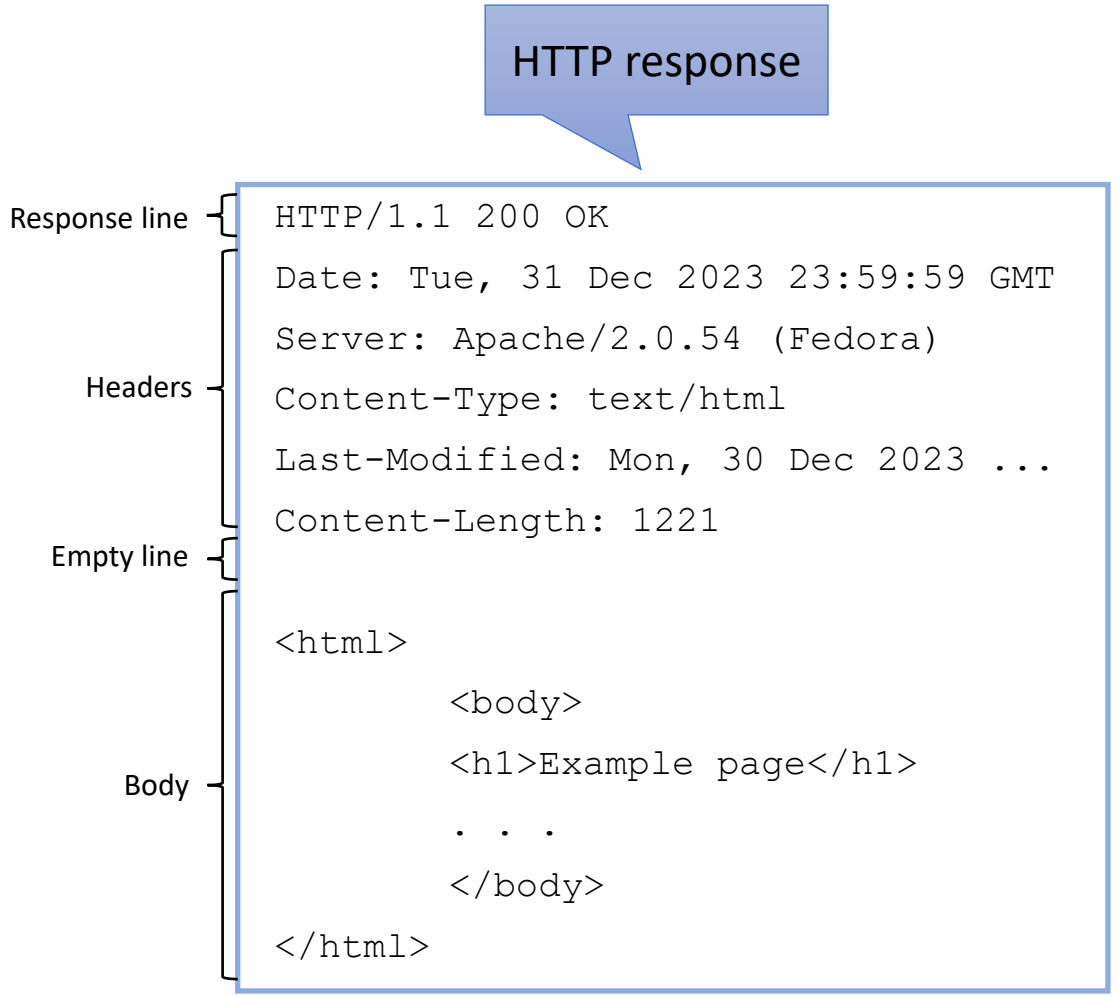


2. HTTP

- 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

- Example of request-response:



2. HTTP

- **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

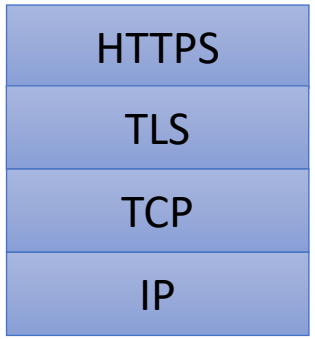


Table of contents

3. REST services

- **REST** (REpresentational State Transfer) is an architectural style for designing distributed services
  - REST is a very popular way for creating web services
- REST is built on top of HTTP, and therefore, it follows a client-server architecture
  - The service server handles a set of resources, listening for incoming requests made by clients
  - Each resource is identified uniquely using URLs known as **endpoints**
  - Each resource has a representation, which is a machine-readable explanation of the current state of a resource. We use a data-interchange format for defining representations, such as **JSON**, YAML, or XML

3. REST services

- We can use the HTTP methods (the so-called verbs) to map REST actions
  - The following table summarizes the HTTP methods used to create REST services:

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 services

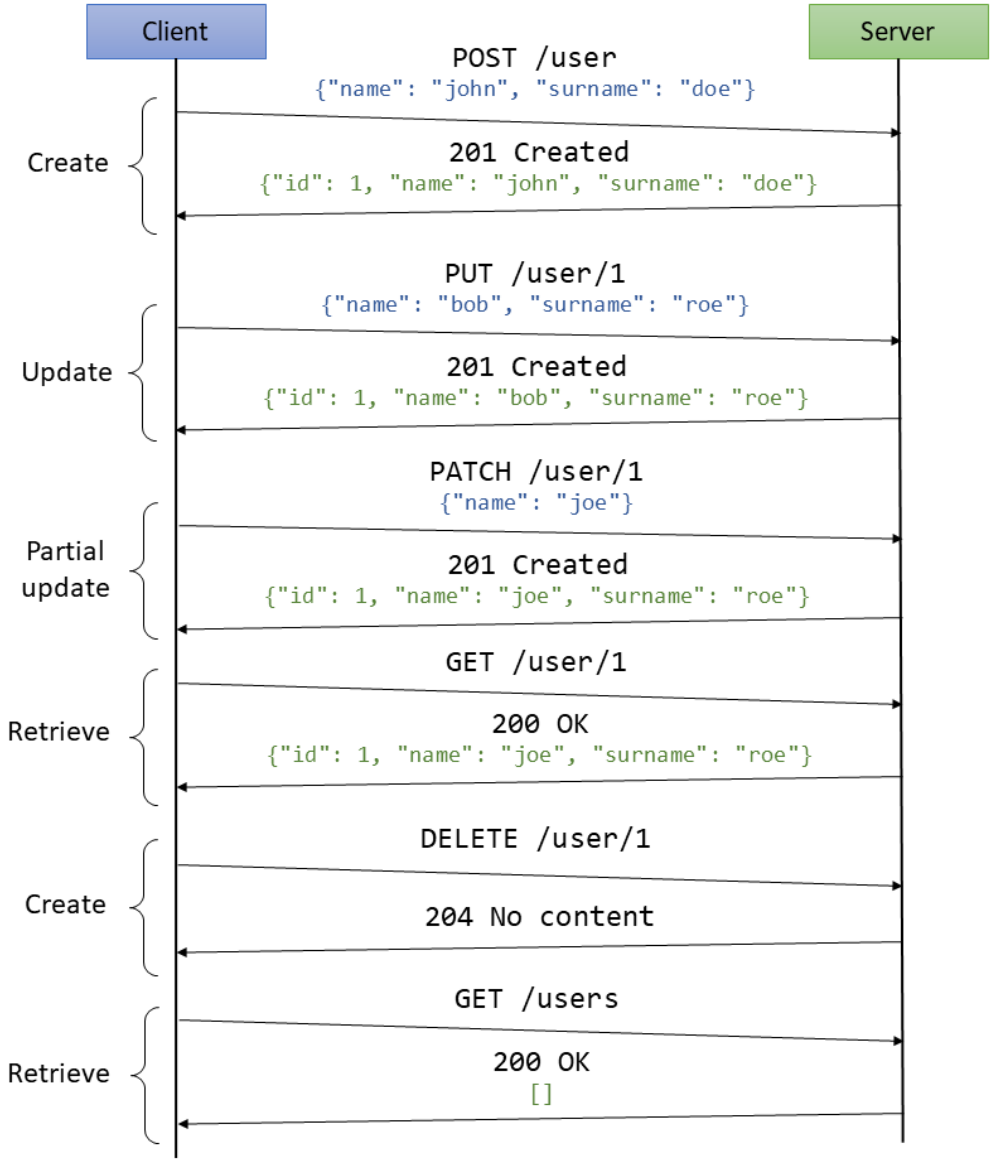
- Finally, we use the HTTP status codes to identify the response associated with REST actions

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 or PUT 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 services

- The following figure shows a sequence of requests and responses of an example REST service that uses different HTTP methods and response codes



3. REST services - JSON

- JSON (JavaScript Object Notation) is a lightweight data-interchange format
  - It is one of the most popular data formats for web services nowadays
  - The JSON format has been defined as an open standard
  - Example of JSON: →

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

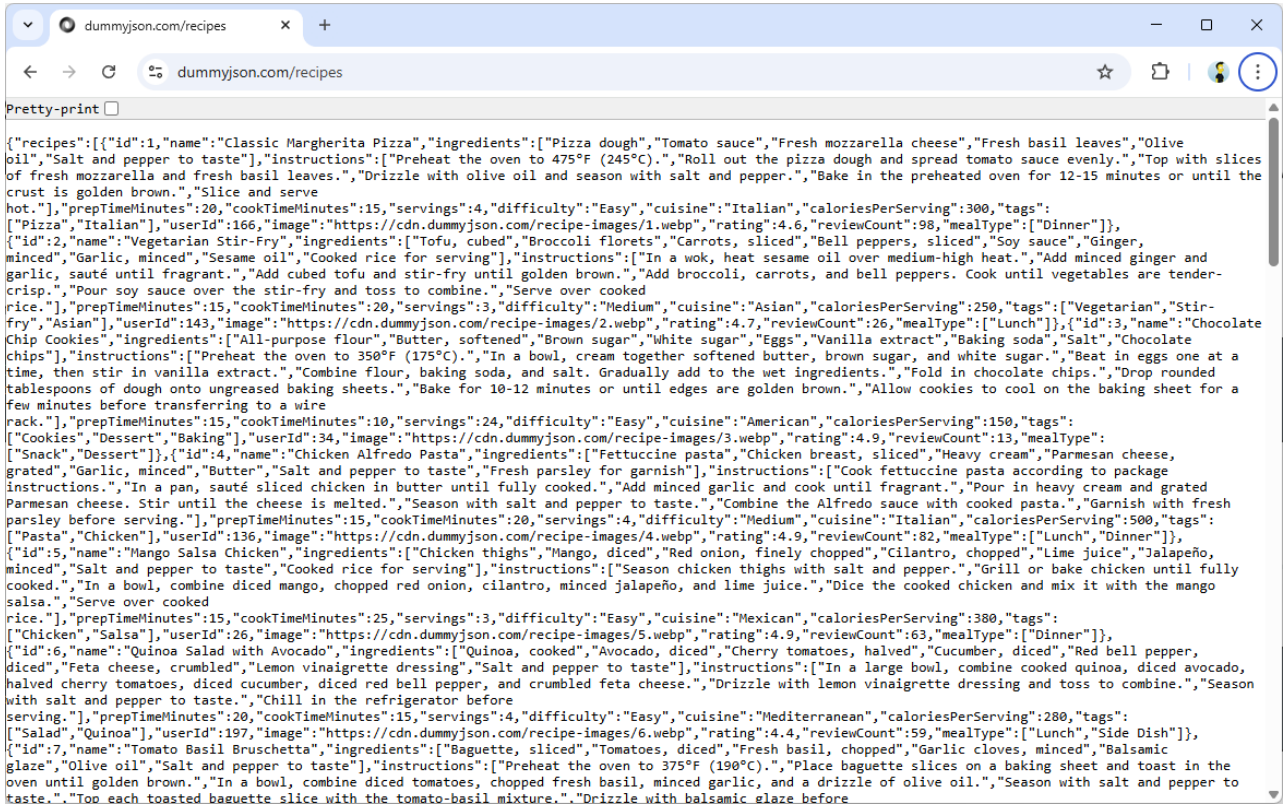
3. REST services - JSON

- JSON data can be:
  - 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 services - 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



DummyJSON is a public REST service for testing and prototyping

3. REST services - 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/

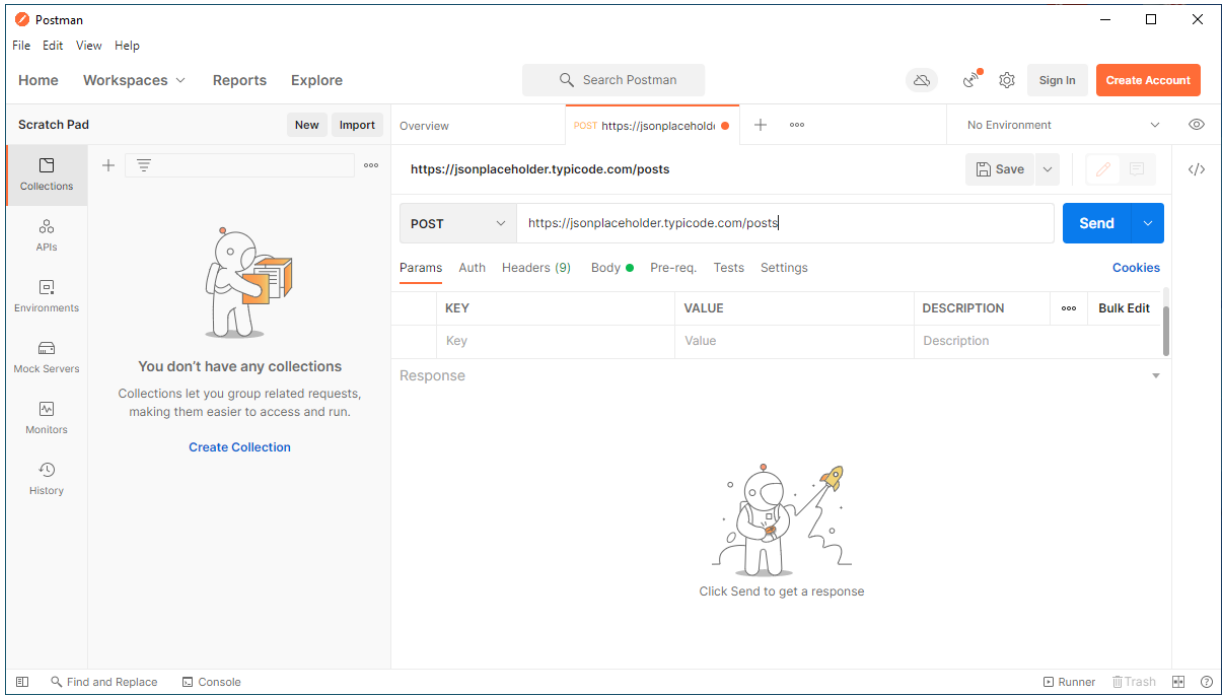


Table of contents

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 (in 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**. For that, first we need to include the following dependencies:

build.gradle.kts (app)

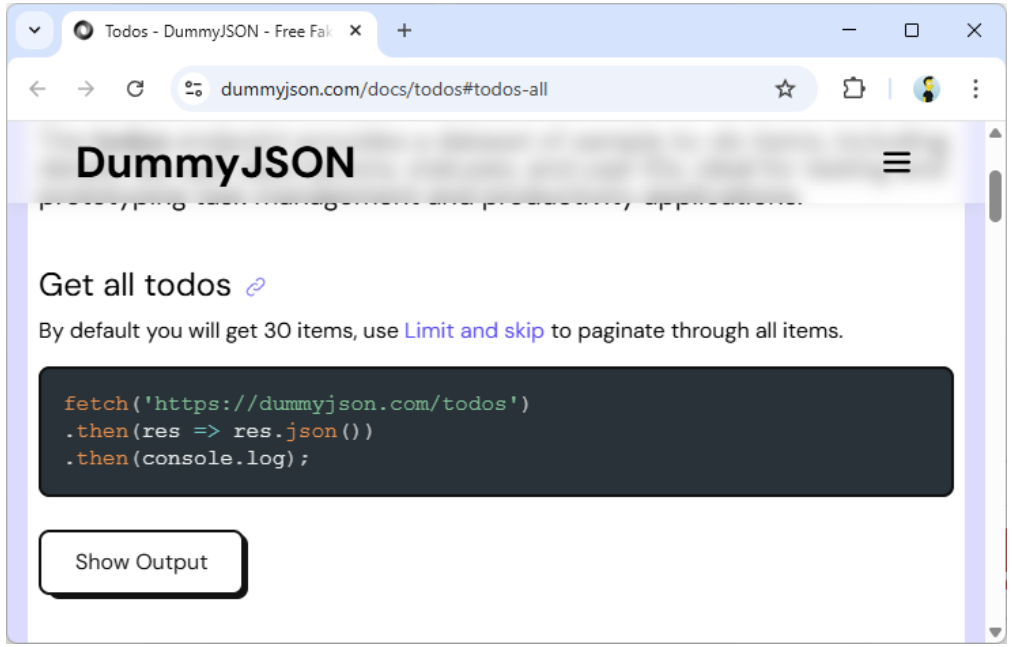
```
implementation(libs.retrofit)
implementation(libs.converter.gson)
implementation(libs.kotlinx.coroutines.android)
```

libs.version.toml

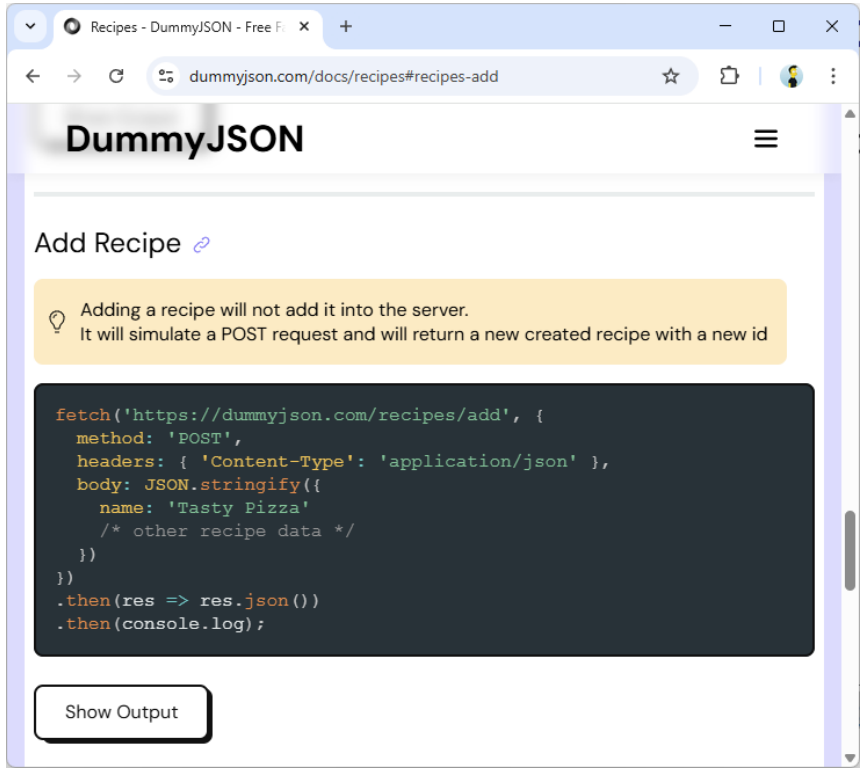
```
[versions]
retrofit = "2.11.0"
converterGson = "2.11.0"
kotlinxCoroutinesAndroid = "1.7.3"

[libraries]
retrofit = { module = "com.squareup.retrofit2:retrofit", version.ref = "retrofit" }
converter-gson = { module = "com.squareup.retrofit2:converter-gson", version.ref =
"converterGson" }
kotlinx-coroutines-android = { module = "org.jetbrains.kotlinx:kotlinx-coroutines-
android", version.ref = "kotlinxCoroutinesAndroid" }
```

4. REST clients in Android - Retrofit

- The following sample app makes GET and POST requests to the REST service DummyJSON using Retrofit



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



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

4. REST clients in Android - Retrofit


```kotlin
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,
)
```

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

```kotlin
interface DummyJsonService {

    @GET("todos")
    suspend fun getTodos(): Response<Todos>

    @POST("recipes/add")
    suspend fun addRecipes(@Body recipe: Recipe): Response<Recipe>

}
```

We need to define an interface for each endpoint we want to use (without the base URL)

```kotlin
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

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!!
                }
            } catch (e: Exception) {
                _toastMessage.value = e.message
            } finally {
                _isLoading.value = false
            }
        }
    }

    fun addRecipe(recipe: Recipe) {
        viewModelScope.launch {
            try {
                val response = DummyJsonClient.apiService.addRecipes(recipe)
                _toastMessage.value = response.code().toString() + " " + response.message()
            } catch (e: Exception) {
                _toastMessage.value = e.message
            }
        }
    }

}
```

We use the previous API service from a `ViewMovel`

4. REST clients in Android - Retrofit


```
@Composable
fun UserListScreen(viewModel: RestViewModel = viewModel()) {
    val context = LocalContext.current
    val todos by viewModel.todos.collectAsState()
    val isLoading by viewModel.isLoading.collectAsState()
    var showDialog by remember { mutableStateOf(false) }
    val toastMessage by viewModel.toastMessage.collectAsState()

    Scaffold(
        floatingActionButton = {
            FloatingActionButton(onClick = { showDialog = true }) {
                Icon(Icons.Default.Add, contentDescription = stringResource(R.string.add))
            }
        }) { padding ->
        Column(
            modifier = Modifier
                .fillMaxSize()
                .padding(padding),
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center
        ) {
            if (isLoading) {
                CircularProgressIndicator()
            } else {
                LazyColumn {
                    items(todos) { todo ->
                        TodoItem(todo = todo)
                    }
                }
            }
        }

        // ...
    }
}
```

Finally, we observe the changed in the `ViewModel` and display the responses in the UI

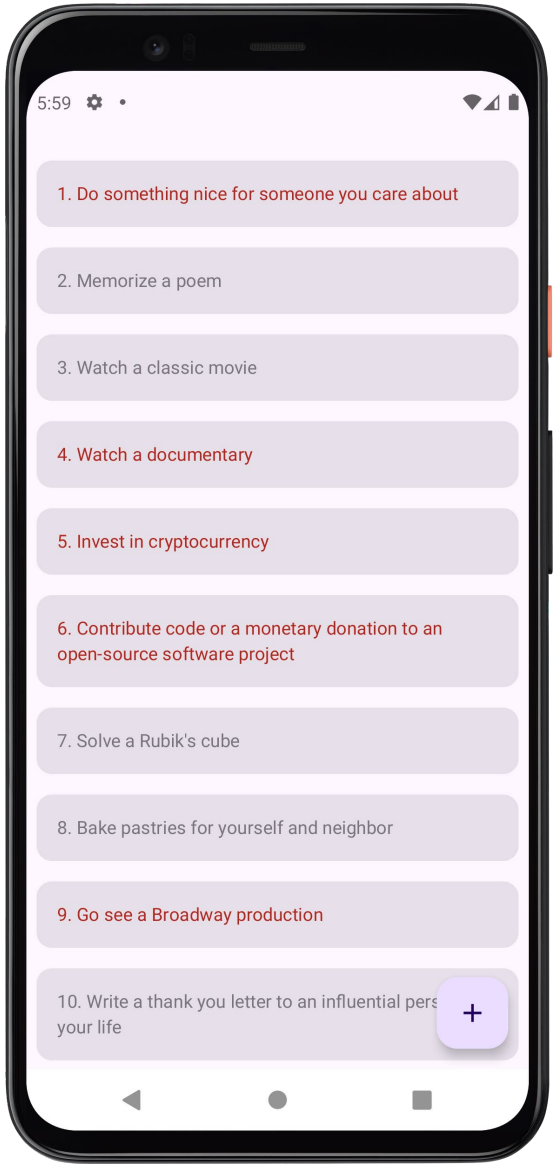


Table of contents

5. Cloud functions

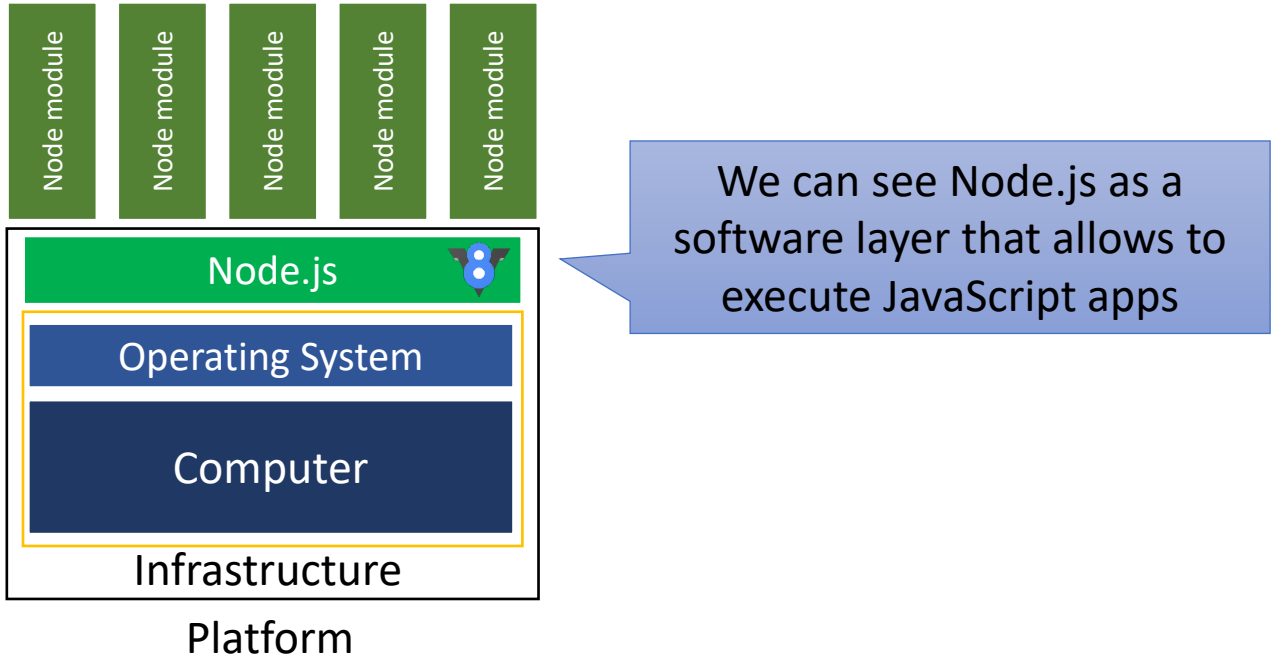
- **Cloud Functions** is a *serverless* framework provided by Firebase to run backend code in response to events triggered by background events, such as HTTP requests
  - Serverless is a cloud computing model where cloud providers (such as GCP) manage the infrastructure needed to execute code, allowing developers to focus solely on writing and deploying functions or blocks of code
- We can use JavaScript, TypeScript or Python code to implement Cloud Functions
  - In this course, we see a very basic Cloud Function example implemented in JavaScript

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

5. Cloud functions - Node.js

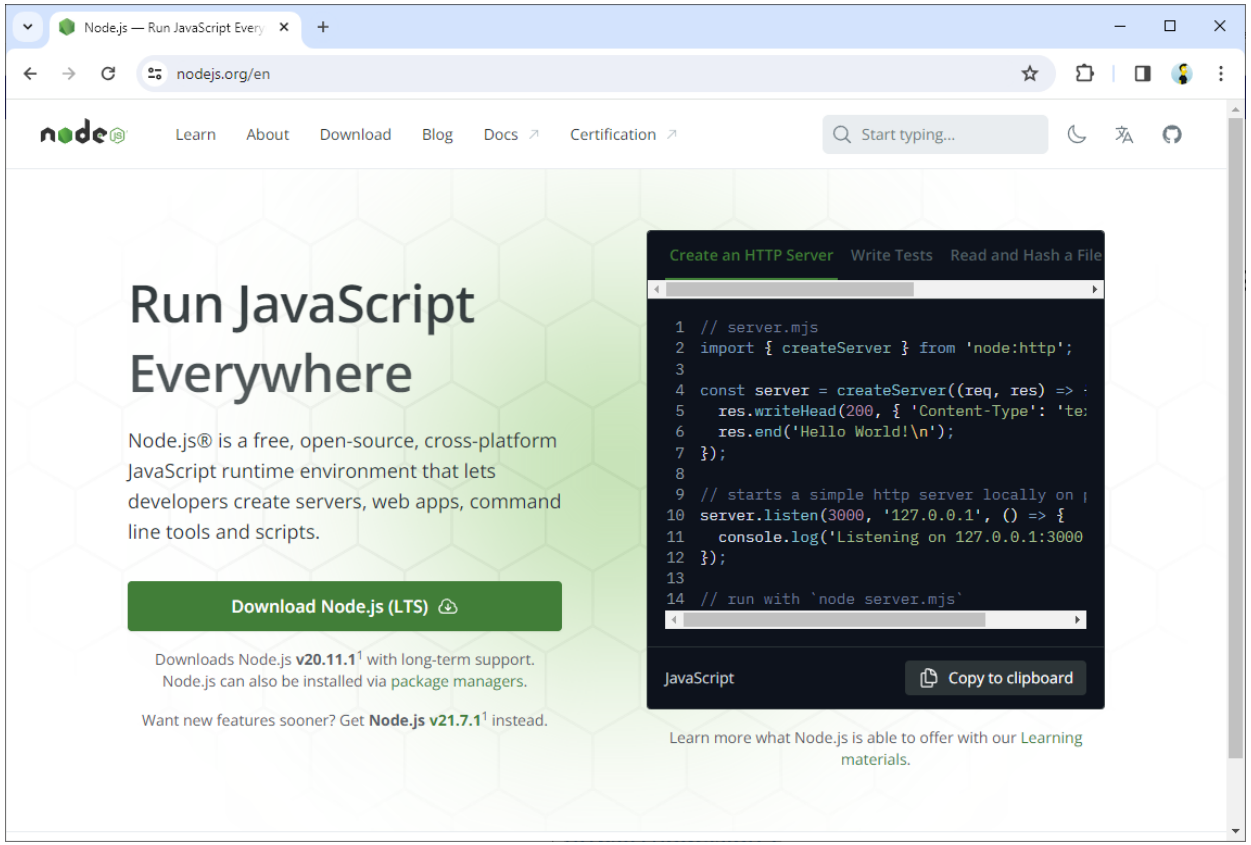
- Node.js is an open-source, cross-platform JavaScript runtime environment that enables the execution of JavaScript code outside a web browser
- Node.js runs on the V8 JavaScript engine
  - V8 is an open-source JavaScript engine developed by the Chromium Project

https://nodejs.org/



5. Cloud functions - Node.js

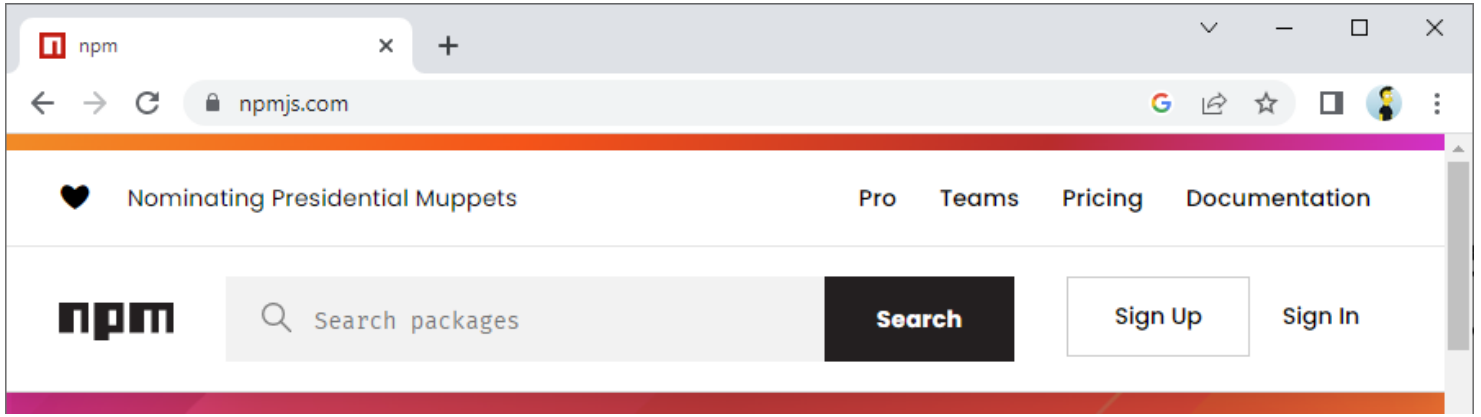
- We can download the Node.js installer from its website:



https://nodejs.org/

5. Cloud functions - 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
v20.11.1

> npm --version
10.5.0
```

https://www.npmjs.com/

5. Cloud functions - Hello world

- The procedure to create a cloud function is the following:
  1. Create a Firebase project
     - We already have Firebase projects in this course (**uc3m-it-2025-16504-g**-lab**)
  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. Cloud functions - Hello world

2. Set up Firebase CLI:

```
> npm install -g firebase-tools

changed 644 packages in 27s

> firebase --version
13.33.0
```

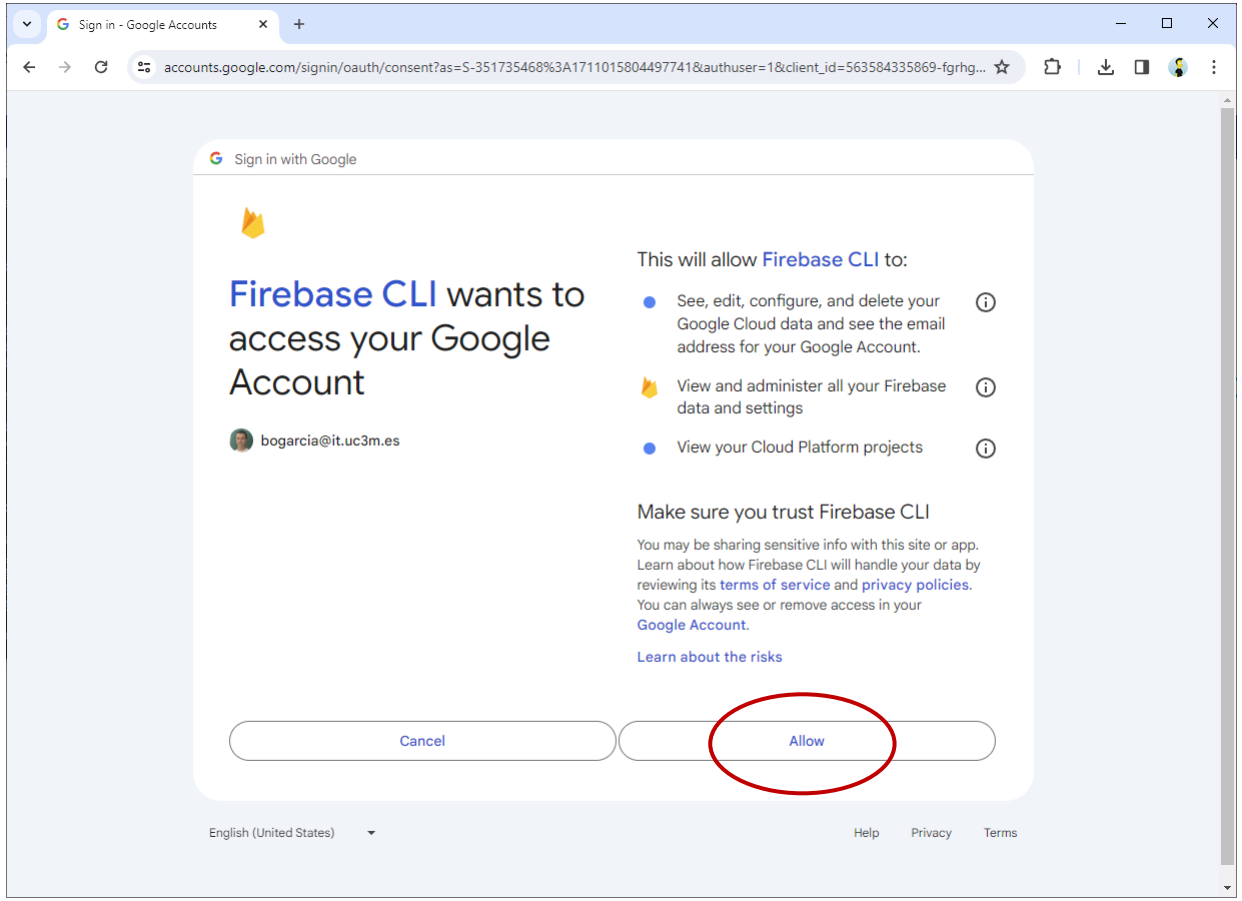
We use npm to install Firebase CLI

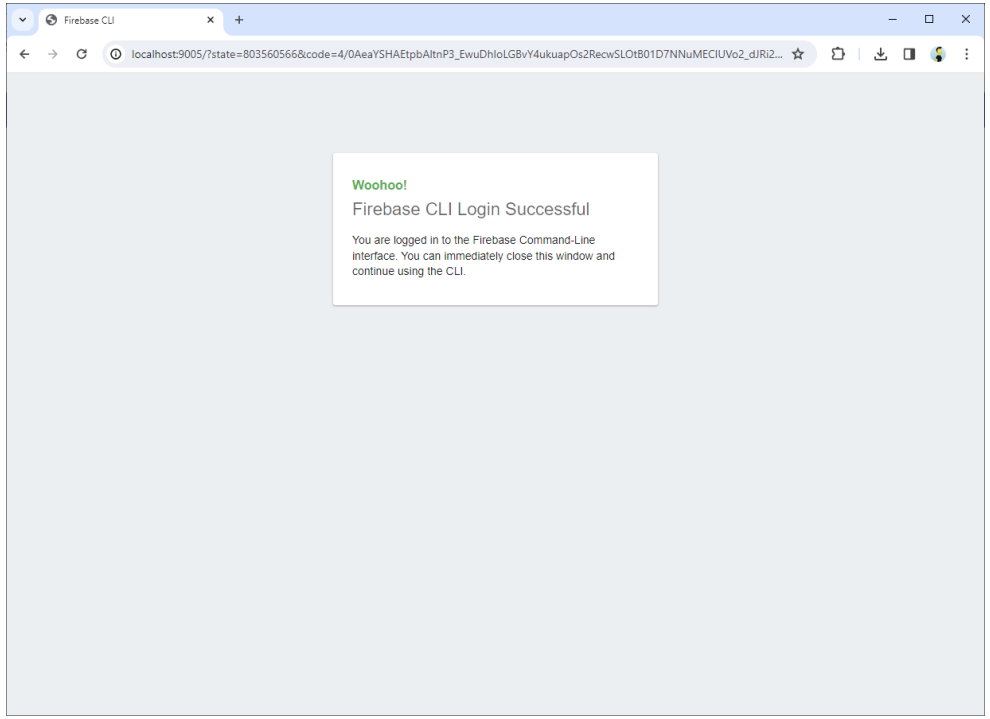
3. Login in Firebase:

```
> 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. Cloud functions - Hello world

3. Login in Firebase:





5. Cloud functions - Hello world

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

5. Cloud functions - Hello world

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

? 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. Cloud functions - Hello world

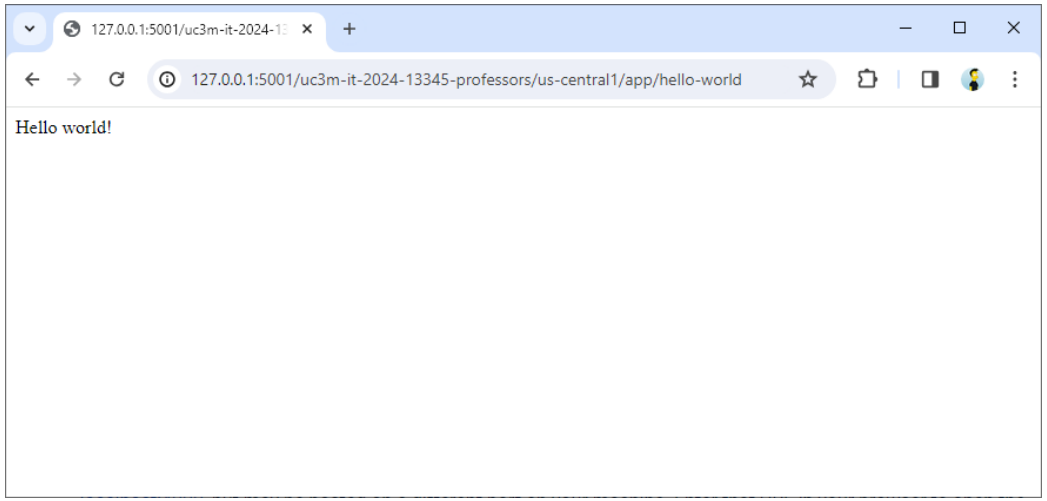
5. Implement the cloud function:
   - You can find a complete project example in GitHub: https://github.com/bonigarcia/cloud-functions-hello-world

```
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. Cloud functions - Hello world

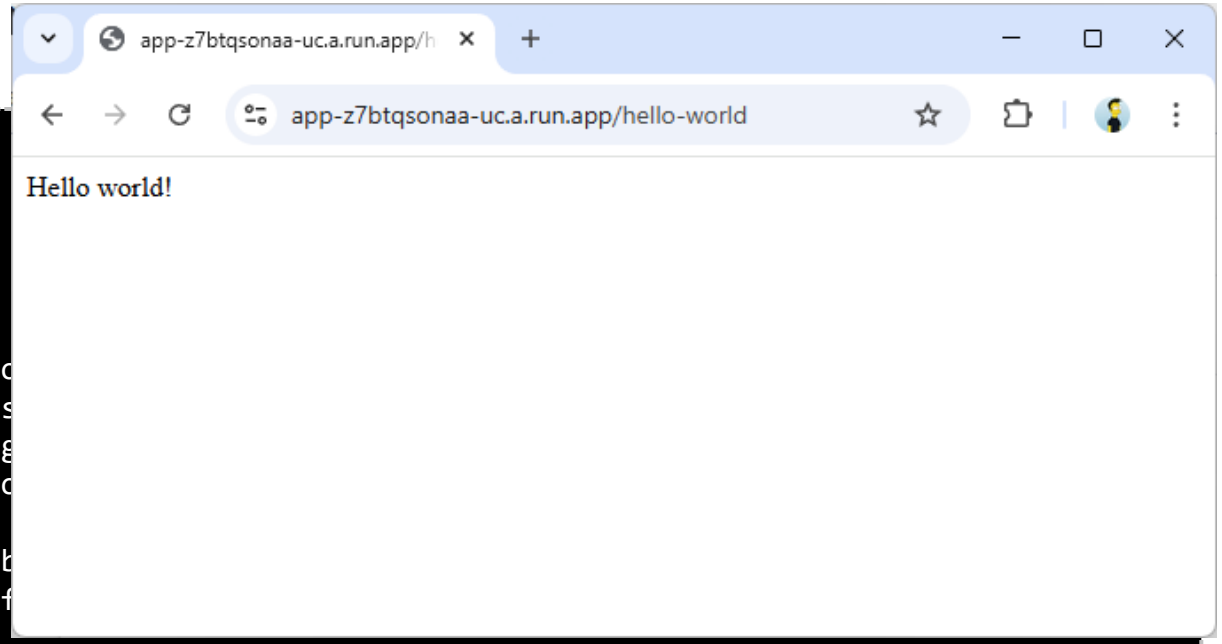
6. Emulate the cloud function locally:

```
> 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. Cloud functions - Hello world

7. Deploy to Firebase:




```
> firebase deploy --only functions

=== Deploying to 'uc3m-it-2024-13345-professors'...

i  deploying functions
i  functions: preparing codebase default for deployment
i  functions: ensuring required API cloudfunctions.googleapis.co
i  functions: ensuring required API cloudbuild.googleapis.com is
i  artifactregistry: ensuring required API artifactregistry.goog
+  artifactregistry: required API artifactregistry.googleapis.co
+  functions: required API cloudbuild.googleapis.com is enabled
+  functions: required API cloudfunctions.googleapis.com is enab
i  functions: Loading and analyzing source code for codebase def
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-2024-13345-professors.cloudfunctions.net/app
i  functions: cleaning up build files...

+  Deploy complete!

Project Console: https://console.firebase.google.com/project/uc3m-it-2025-13345-professors/overview
```

Table of contents

6. Takeaways

- A web service is a distributed software system designed to allow different software to interact built on the top of HTTP
- REST is a popular architectural style for implementing web services
- JSON is a lightweight data-interchange format very popular for data exchange in REST services
- We can implement a REST client in an Android app using an existing library such as Retrofit
- Cloud Functions is a serverless framework provided by Firebase that allows us to implement REST services in an easy way