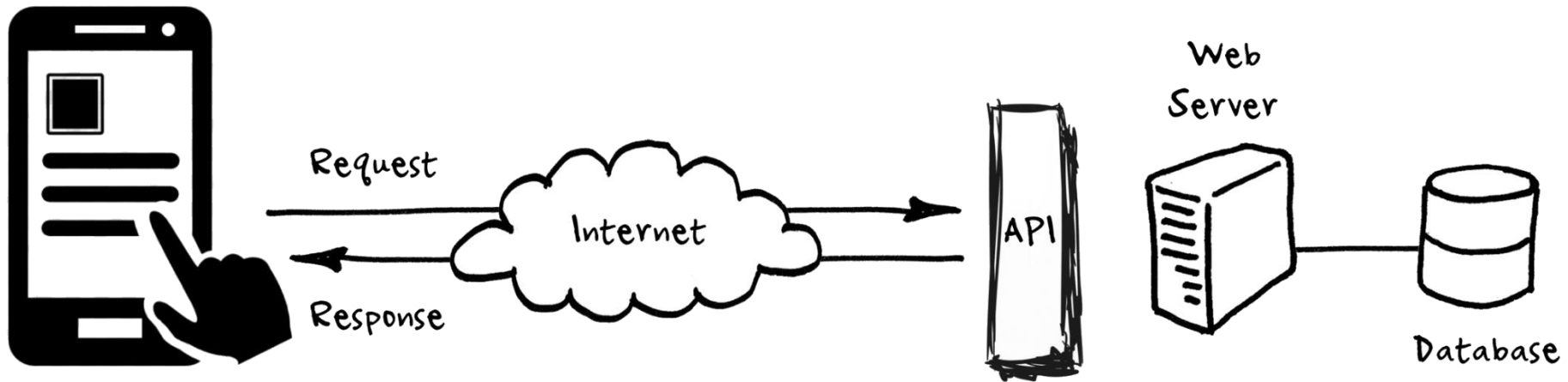


Calling Web API using Retrofit & Coroutines





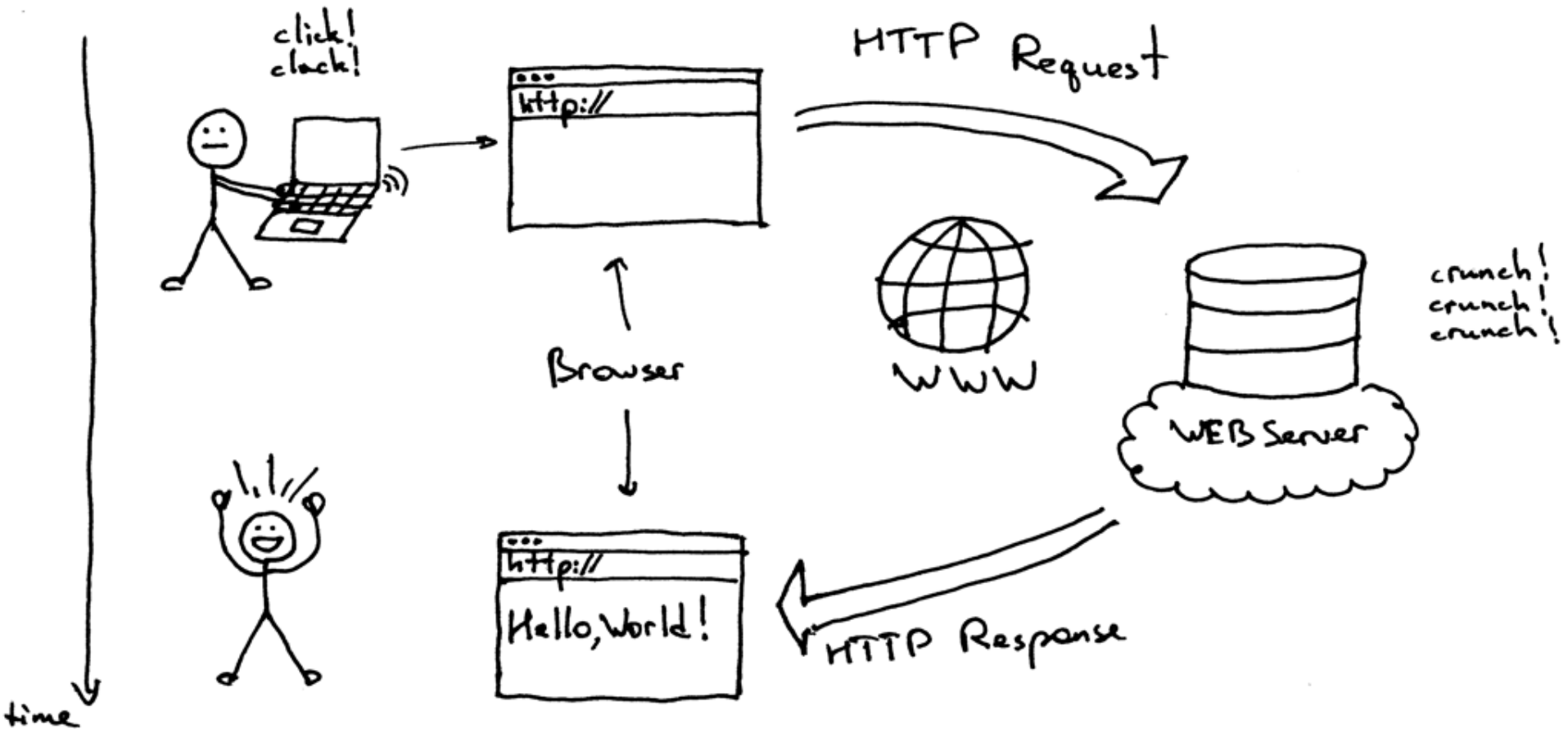
Web and HTTP



What is Web?

- Web = **global distributed system of interlinked resources** accessed over the Internet using the **HTTP protocol**
 - Consists of set of **resources** located on different servers:
HTML pages, images, videos and other resources
 - Resources have unique **URL** (Uniform Resource Locator) address
 - Accessed through standard **HTTP** protocol
- The Web has a Client/Server architecture:
 - **Web browser / Mobile App** requests resources (using HTTP protocol) and displays them
 - **Web server** sends resources in response to requests (using HTTP protocol)

How the Web Works?



Uniform Resource Locator (URL)

`http://www.qu.edu.qa:80/cse/logo.gif`

protocol host name Port Url Path

- URL is a formatted string, consisting of:
 - **Protocol** for communicating with the server (e.g., http, https, ...)
 - **Name of the server or IP** address plus port (e.g. `qu.edu.qa:80`, `localhost:8080`)
 - **Path of a resource** (e.g. `/ceng/index.html`)
 - **Parameters** aka **Query String** (optional), e.g.
`https://www.google.com/search?q=qatar%20university`

Web API (aka Web Services)

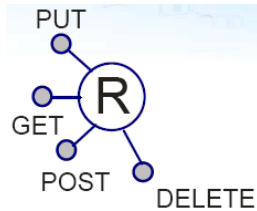


What is a Web API?

- Web API = Web accessible Application Programming Interface accessible via HTTP to allow programmatic access to applications
 - Also known as Web Services
 - Can be accessed by a broad range of clients including browsers and mobile devices
- Web API is a web service that accepts requests and returns **structured data** (JSON in most cases)
 - Programmatically accessible at a particular URL
 - You can think of it as a Web page returning JSON instead of HTML
- Major goal = **interoperability between heterogeneous systems**



Web Services Principles



- **Resources have unique address (nouns)** i.e., a **URI**
e.g., `http://example.com/customers/123`
- **Can use a Uniform Interface (verbs)** to access them:
 - HTTP verbs: GET, POST, PUT, and DELETE
- **Resource has representation(s) (data format)**
 - A resource can be in a variety of data formats: **JSON**, **XML**, **RSS**..

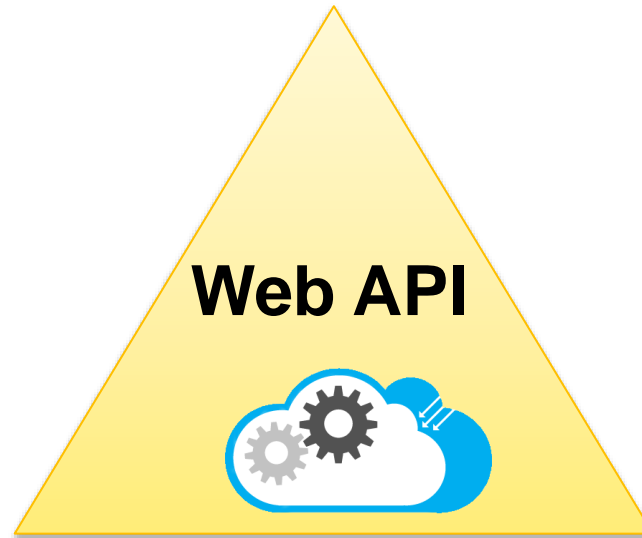
Resources

- The key abstraction in REST is a **resource**
- A resource is a conceptual mapping to a set of entities
 - Any **information that can be named can be a resource**: a document or image, a temporal service (e.g. "today's weather in Doha"), a collection of books and their authors, and so on

Web API Main Concepts

Nouns (Resources)

e.g., <http://example.com/employees/12345>



Verbs

e.g., GET, POST

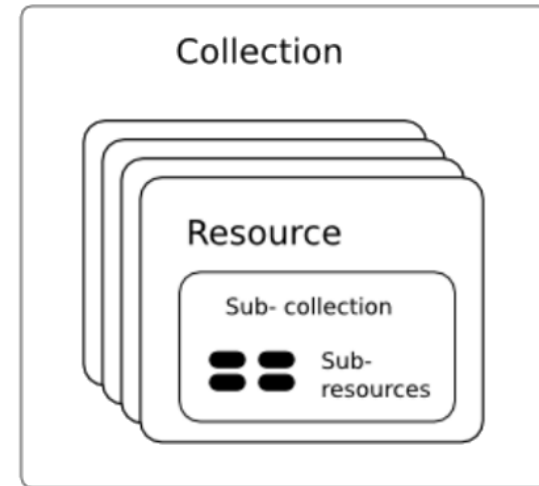
Representations

e.g., XML, JSON

Naming Resources

- Web API uses URL to identify resources

Often **api** path is used for better organization



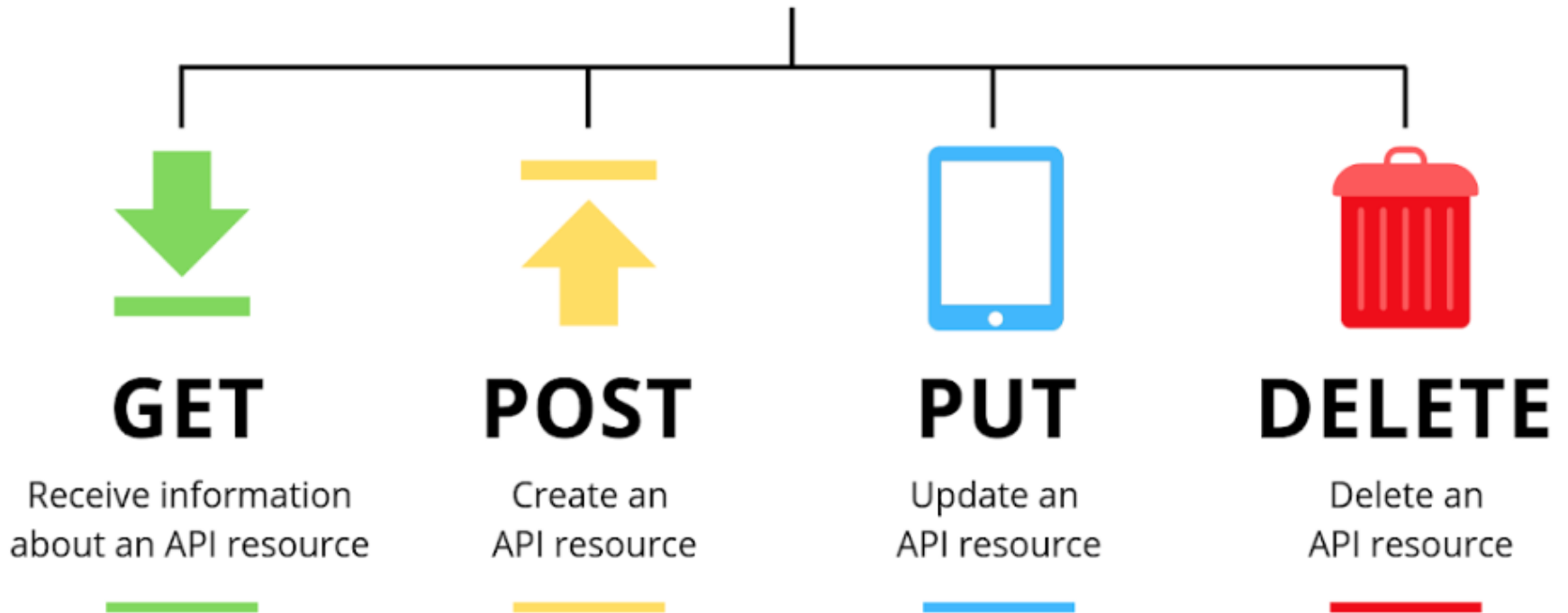
- <http://localhost/api/books/>
 - <http://localhost/api/books/ISBN-0011>
 - <http://localhost/api/books/ISBN-0011/authors>

 - <http://localhost/api/classes>
 - <http://localhost/api/classes/cmcs356>
 - <http://localhost/api/classes/cs356/students>
- As you traverse the **path** from more generic to more specific, you are navigating the data

HTTP Verbs

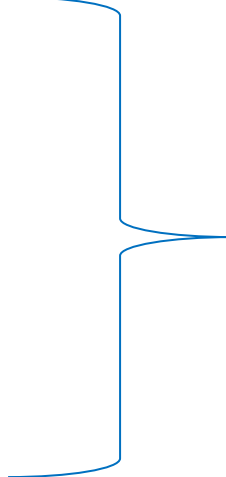
HTTP Verbs represent the **actions** to be performed on resources

REST API Methods



CRUD (Create, Read, Update and Delete) Operations and their Mapping to HTTP Verbs

- **GET** - Read a resource
 - **GET** /books - Retrieve all books
 - **GET** /books/:id - Retrieve a particular book
- **POST** - Create a new resource
 - **POST** /books - Create a new book
- **PUT** - Update a resource
 - **PUT** /books/:id - Update a book
- **Delete** – Delete a resource
 - **DELETE** /books/:id - Delete a book



The resource data (e.g., book details) are placed in the **body** of the request

Example 2 - Task Service API

Task	Method	Path
Create a new task	POST	/tasks
Delete an existing task	DELETE	/tasks/{id}
Get a specific task	GET	/tasks/{id}
Search for tasks	GET	/tasks
Update an existing task	PUT	/tasks/{id}

Representations

Two main formats:

- **JSON**

```
{  
  code: 'cmp123',  
  name: 'Web Development'  
}
```

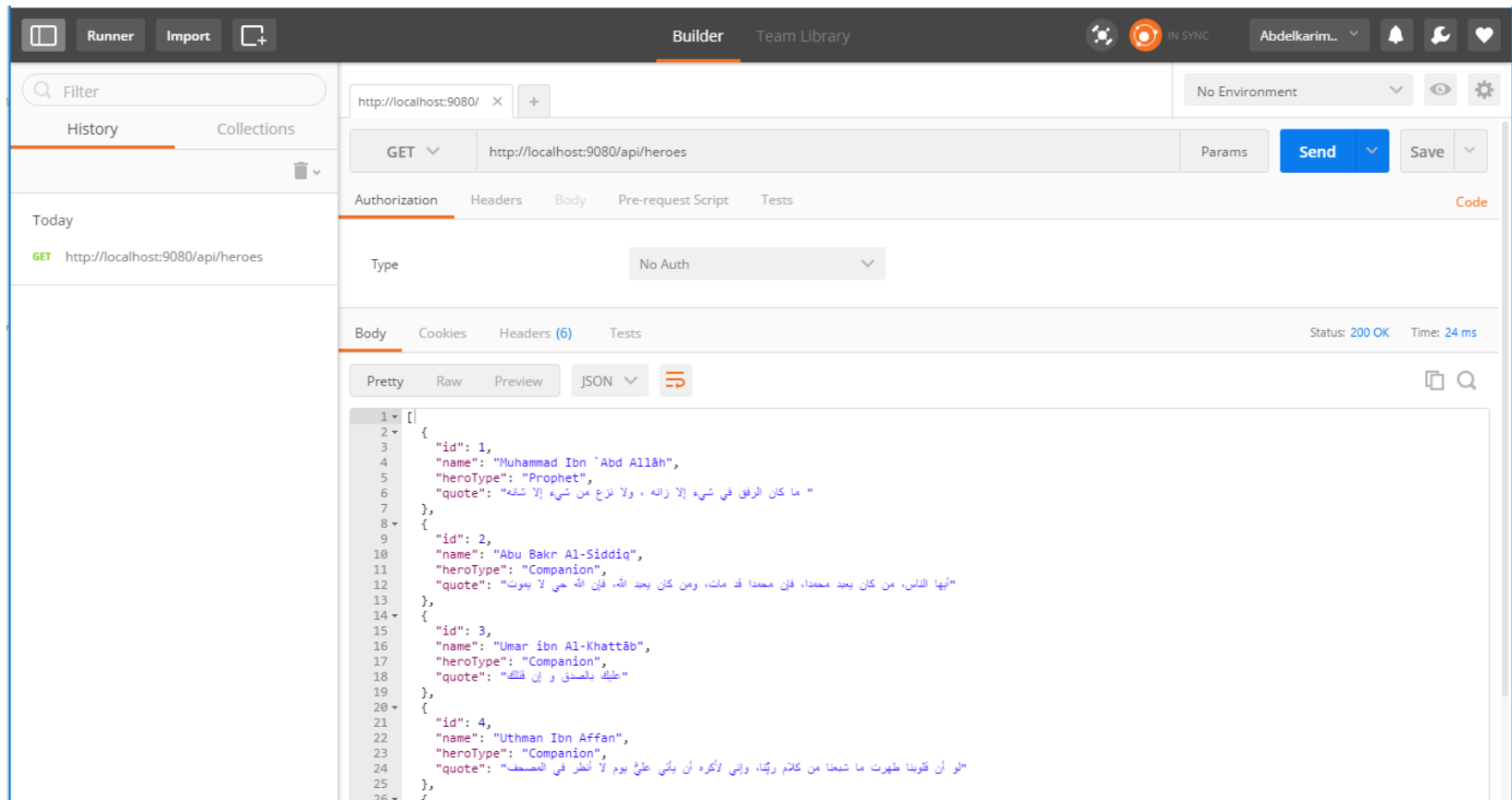
- **XML**

```
<course>  
  <code>cmp123</code>  
  <name>Web Development</name>  
</course>
```

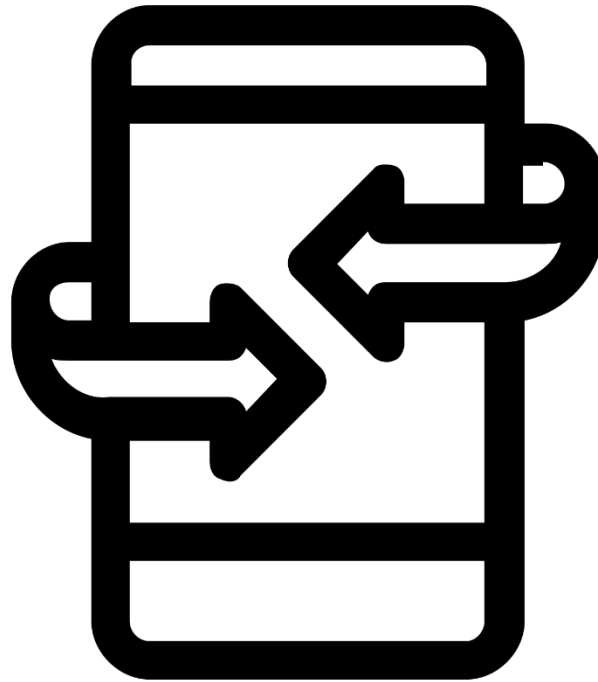
Testing Web API

- Using Postman to test Web API

<https://www.postman.com/downloads/>

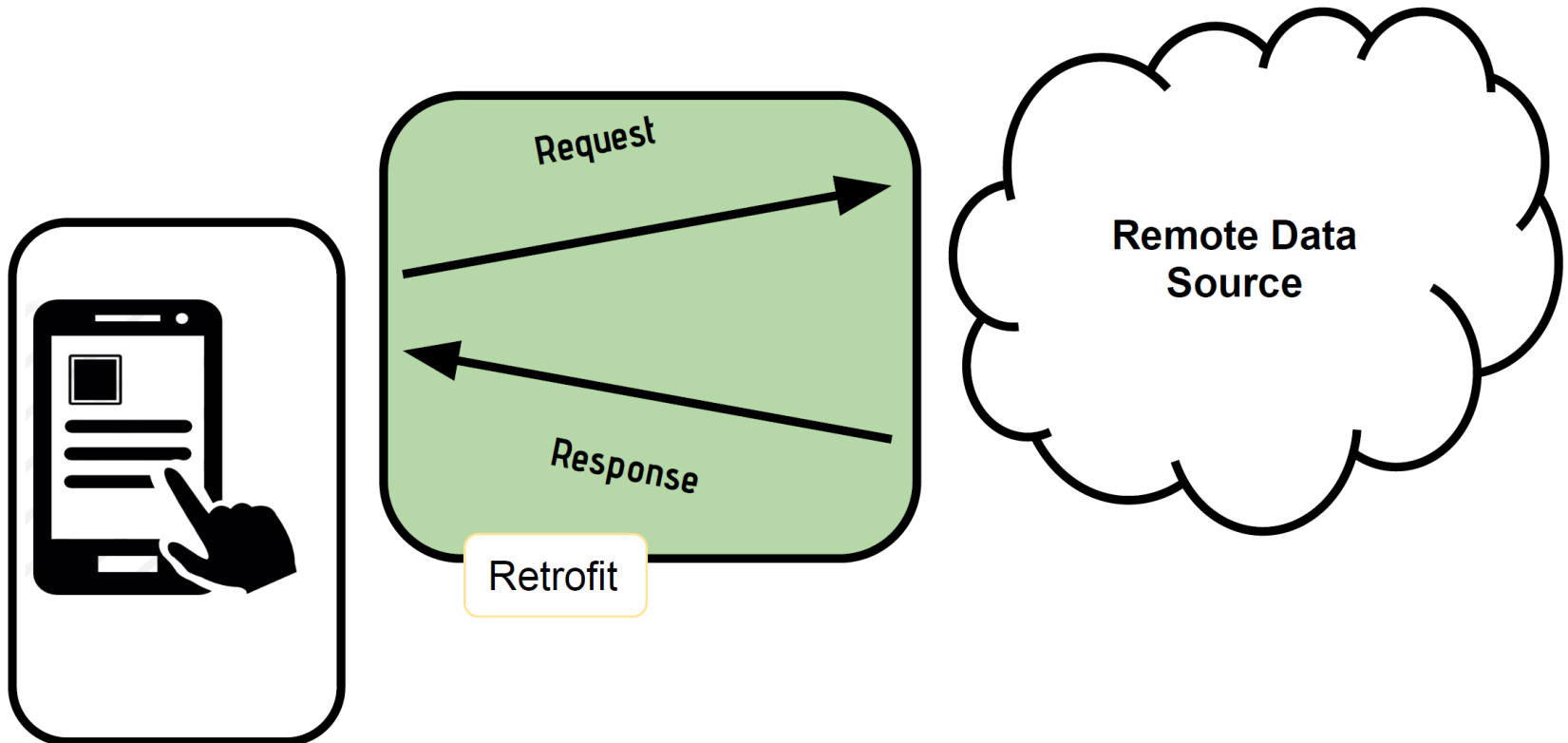


Retrofit



Retrofit Role

- Retrofit provides HTTP client library for Android app to call a remote Web API



Retrofit Programming Steps

3 Steps to use it Retrofit:

- 1) Define **Serializable Data Classes** for input/output objects used when interacting with the Web API
- 2) Define the **Service API** using a Kotlin interface
 - Define how **requests are created** and sent and how their **responses are read and parsed**
 - Method and parameter annotation customize service requests
- 2) Use **Retrofit.Builder** to generate a class to call the remote Web API

1. Define Serializable Data Classes for input/output objects used when interacting with the Web API

```
@Serializable
data class Country (
    val cioc: String = "",
    val name: String,
    val capital: String,
    val region: String,
    val subregion: String,
    val population: Long,
    val area: Double = 0.0,
    val flag: String
)
```

2. Define Service API

```
interface CountryService {
```

```
    @GET("countries")
```

```
    suspend fun getCountries() : List<Country>
```

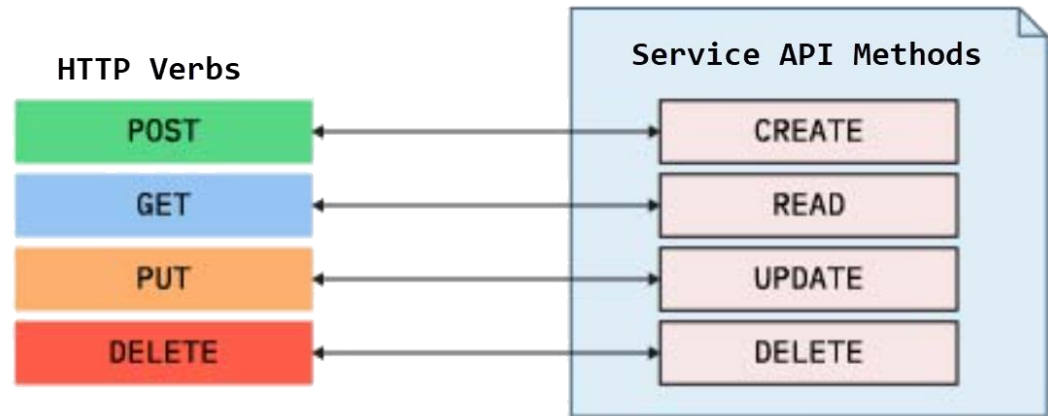
```
    @GET("countries/{name}")
```

```
    suspend fun getCountry(@Path("name") name: String) : Country
```

```
    @POST("countries")
```

```
    suspend fun addCountry(@Body country: Country)
```

```
}
```



3. Use Retrofit.Builder to generate a class to call the remove Web API

```
private const val BASE_URL = "https://restcountries.eu/rest/v2/"
private val contentType = "application/json".toMediaType()

val jsonConverterFactory = Json { ignoreUnknownKeys = true
    coerceInputValues = true }.asConverterFactory(contentType)

val countryService by lazy {
    Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(jsonConverterFactory)
        .build()
        .create(CountryService::class.java)
}
```

Method Annotations



HTTP GET



HTTP POST



HTTP PUT



HTTP
DELETE

- Service Interface methods are annotated based on:
 - **HTTP Verb** – GET, POST, PUT, DELETE used to access the service
 - **URL Path** – e.g., /users
- Method annotation **maps** an **HTTP Verb** (e.g., GET or POST) + a **URI Path** to a **method**
 - E.g., getCountry method is mapped to **Get** verb and /countries/{name} Url path

```
@GET("countries/{name}")  
suspend fun getCountry(@Path("name") name: String) : Country
```

Path Parameters

- Named **path parameters** can be added to the URL path E.g., **/students/{id}**
 - E.g., if you have the path **/students/{id}**, then the “id” parameter is available to the method using **@Path(“id”)**
studentId: Int

```
@GET("/students/{id}")  
suspend fun getStudents(@Path("id") studentId: Int) : Student
```


Query Parameters

- Named **query parameters** can be added to the URL path after a **?** E.g., **/posts?sortBy=createdOnDate**
- Query parameters are often used for **optional** parameters (e.g., optionally specifying the property to be used to sort of results)
- **@Query** annotation is used to map method parameters to for each query parameter in the URL path
 - If you have the path **/posts?sortBy=createdOnDate**, then the “**sortBy**” query parameter is available to the method using **@Query("sortBy") sortBy: String**

```
@GET("posts")  
suspend fun getPosts(@Query("sortBy") sortBy: String) : List<Post>
```

Working with a Request Body

- Annotate the method parameter with **@Body** annotation to send the method parameter as request body in the HTTP request

```
@POST("countries")  
suspend fun addCountry(@Body country: Country)
```