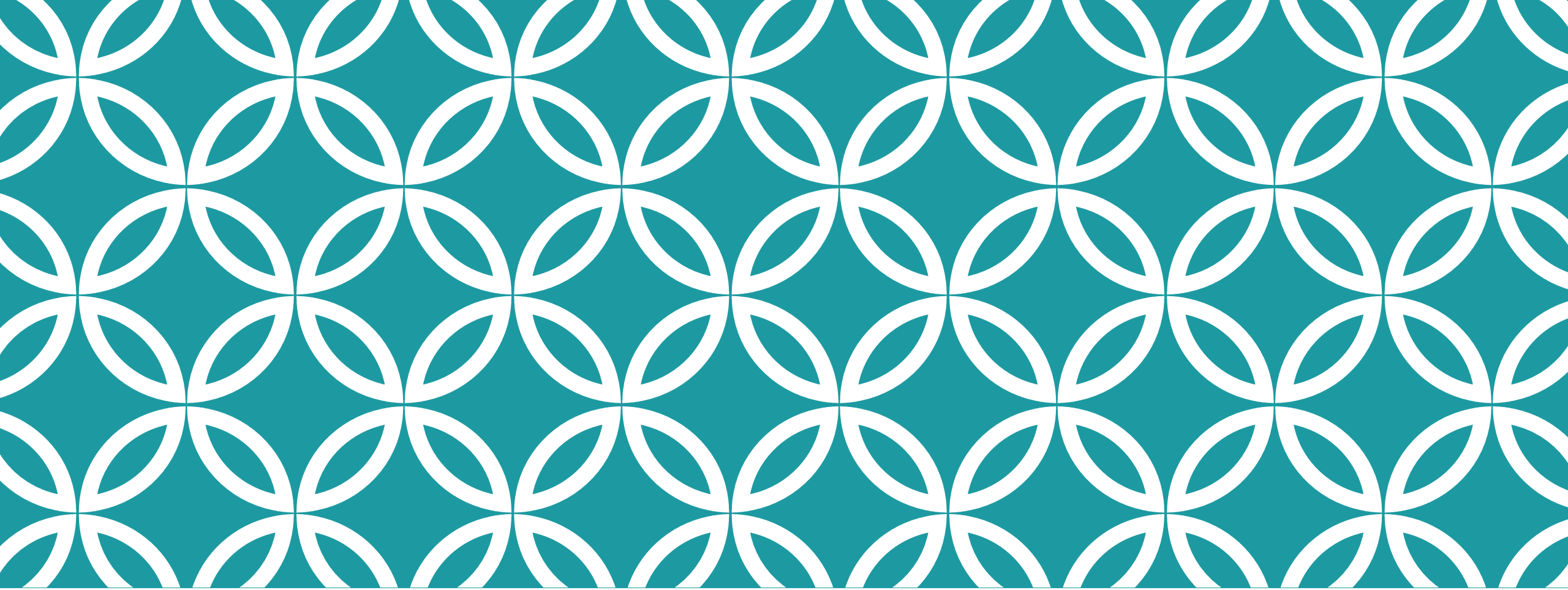
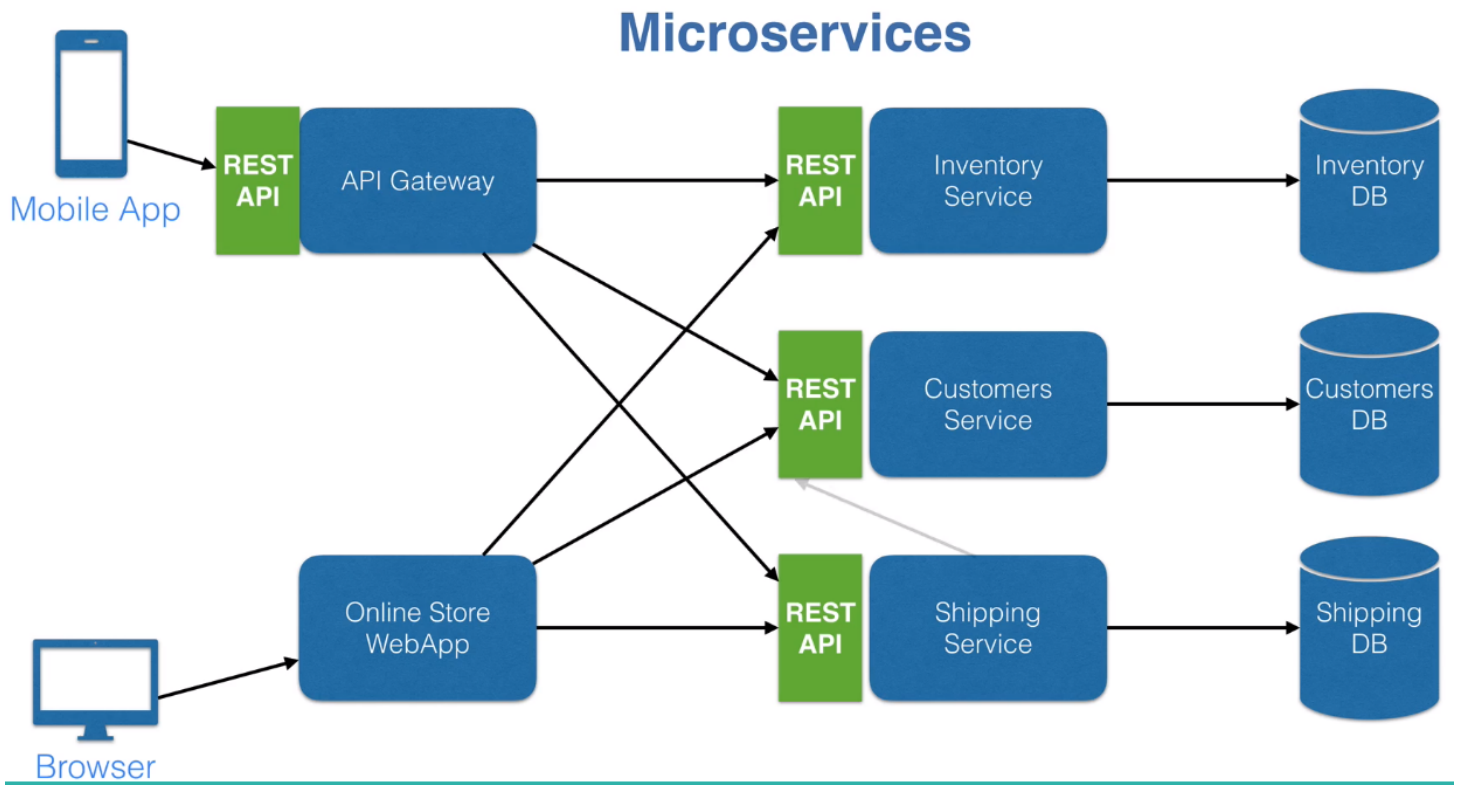


REST-assured

11/16/2021

WHAT IS MICROSERVICES?

REST-assured



WHAT IS REST?

REST-assured

REST stands for **Re**presentational **S**tate **T**ransfer.

Basic principle of rest

☑ **Resources** expressed as readable URLs

http:myapp.com/**api/users/sarah**

resource

☑ **Representations** transferred as JSON or XML

```
{
  "id": 101,
  "firstName":"Sarah",
  "lastName":"Smith"
}
```

☑ **Messages** use standard HTTP methods (e.g. GET, POST, PUT, DELETE)

☑ **Stateless** interactions

REST METHODS

GET retrieves information

- ☑ "READ"
- ☑ Idempotent

```
GET /api/users/101
```

PUT stores an entity

- ☑ "UPDATE"
- ☑ Often used to update an existing record
- ☑ Sometimes used to create a records
- ☑ Should be idempotent

```
PUT /api/users/sarah
```

POST performs an action

- ☑ "WRITE"
- ☑ Often used to create a new record

```
POST /api/users
```

PATCH updates specific fields in an entity

- ☑ "UPDATE"
- ☑ Not idempotent

```
PATCH /api/users/sarah
```

DELETE removes a record

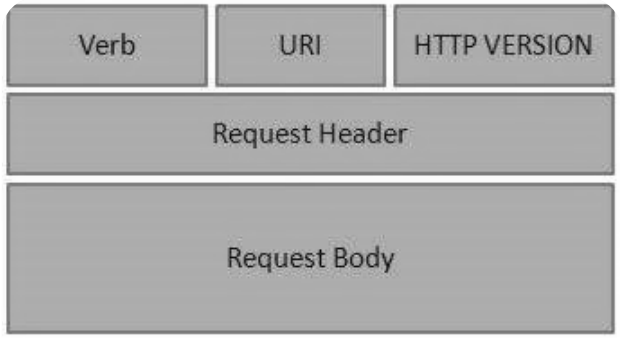
```
DELETE /api/users/sarah
```

REST STATUS CODES

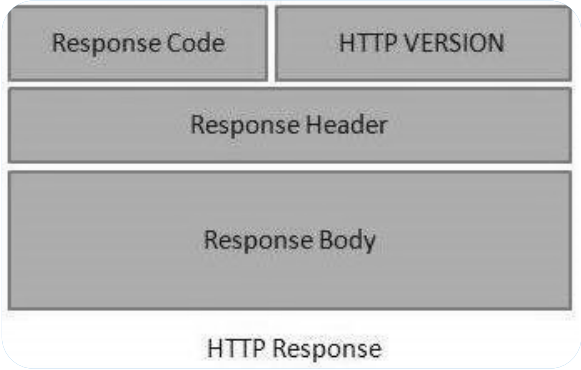
HTTP Verb	CRUD	Response
POST	Create	201 (Created), 409 (Conflict) if resource already exists...
GET	Read	200 (OK), 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	200 (OK). 404 (Not Found), if ID not found or invalid.

RESTFUL WEB SERVICES - MESSAGES

RESTful Web Services make use of HTTP protocols as a medium of communication between client and server. A client sends a message in form of a HTTP Request and the server responds in the form of an HTTP Response.



An HTTP Request has five major parts –
Verb – Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.
URI – Uniform Resource Identifier (URI) to identify the resource on the server.
HTTP Version – Indicates the HTTP version. For example, HTTP v1.1.
Request Header – Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by the client, format of the message body, cache settings, etc.
Request Body – Message content or Resource representation.



An HTTP Response has four major parts –
Status/Response Code – Indicates the Server status for the requested resource. For example, 404 means resource not found and 200 means response is ok.
HTTP Version – Indicates the HTTP version. For example HTTP v1.1.
Response Header – Contains metadata for the HTTP Response message as keyvalue pairs. For example, content length, content type, response date, server type, etc.
Response Body – Response message content or Resource representation.

Query parameters: These are appended at the end of a RESTful API endpoint and can be identified by the question mark in front of them. For example, in the endpoint http://md5.jsontest.com/?text=test, "text" is a query parameter (with value "test").

`http://md5.jsontest.com/?text=testcaseOne`

`http://md5.jsontest.com/?text=testcaseTwo`

Path parameters: These are part of the RESTful API endpoint. For example, in the endpoint we used earlier: http://ergast.com/api/f1/2017/circuits.json, "2017" is a path parameter value.

http://api.zippopotam.us/us/90210

http://api.zippopotam.us/ca/B2A

REST APIS PARAMETERS:

REST-assured

REST Assured is a Java DSL for simplifying testing of REST based services built on top of HTTP Client.

It supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD requests and can be used to validate and verify the response of these requests.

REST Assured can be used to test XML as well as JSON based web services.

REST Assured can be integrated with JUnit and TestNG frameworks for writing test cases for our application. It also supports BBD/Gherkin

REST assured is the support of XML Path and JSON Path syntax to check specific elements of the response data.

REST ASSURED

REST-assured

ABOUT HAMCREST MATCHERS

`equalTo(X)`	Does the object equal X?
`hasItem("Rome")`	Does the collection contain an item "Rome"?
`hasSize(3)`	Does the size of the collection equal 3?
`not(equalTo(X))`	Inverts matcher equalTo()

Express expectations in natural language

http://hamcrest.org/JavaHamcrest/javadoc/1.3/org/hamcrest/Matchers.html

- Get
- Post
- Delete
- Update
- JSON Parsing
- XML Parsing
- E2E flow

CODE WALKTHRO UGH

REST-assured

SPECIFYING REQUEST DATA

HTTP resources

when().get("/x"). ..;

Parameters

given().param("param1", "value1").param("param2", "value2").when().get("/something");

Multi-value parameter

List<String> values = new ArrayList<String>(); values.add("value1");
values.add("value2");

given().param("myList", values). ..

Path parameters

post("/reserve/{hotelId}/{roomNumber}", "My Hotel", 23);

Cookies

given().cookie("username", "John").when().get("/cookie").then().body(equalTo("username"));

Headers

given().header("MyHeader", "Something").and(). ..

given().headers("MyHeader", "Something", "MyOtherHeader", "SomethingElse").and(). ..

Content Type

given().contentType(ContentType.TEXT). ..

given().contentType("application/json"). ..

Request Body

given().body("some body"). .. // Works for POST, PUT and DELETE requests

given().request().body("some body"). .. // More explicit (optional)

VERIFYING RESPONSE DATA

```
get("/x").then().assertThat().statusCode(200). ..
```

Cookies

```
get("/x").then().assertThat().cookie("cookieName", "cookieValue"). ..
get("/x").then().assertThat().cookies("cookieName1", "cookieValue1",
"cookieName2", "cookieValue2"). ..
```

Headers

```
get("/x").then().assertThat().header("headerName", "headerValue"). ..
get("/x").then().assertThat().headers("headerName1", "headerValue1",
"headerName2", "headerValue2"). ..
```

Content Type

```
get("/x").then().assertThat().contentType(ContentType.JSON). ..
```

Request Body

```
get("/x").then().assertThat().body(equalTo("something")). ..
```

Measuring Response Time

```
get("/lotto").timeIn(SECONDS);
```

CONSTRUCT REQUEST OR RESPONSE SPECIFICATI

REST-assured

You can use the builder to construct a request or response specification.

RequestSpecBuilder:

RequestSpecification spec=new RequestSpecBuilder().setBaseUri("http://api.zippopotam.us").build();

given().spec(spec).when().get("/us/90210").then().statusCode(200);

ResponseSpecBuilder:

ResponseSpecification spec=new ResponseSpecBuilder().expectContentType(ContentType.*JSON*).*expectStatusCode(200).build();*

given().when().get("http://api.zippopotam.us/us/90210").then().spec(spec).and().body("country", equalTo("United States"));

ABOUT GPATH

REST-assured

JsonPath is a query language for JSON documents

REST Assured using the GPath implementation

Similar aims and scope as XPath for XML

Documentation and examples:

http://groovy-lang.org/processing-xml.html#_gpath

http://groovy.jmiguel.eu/groovy.codehaus.org/GPath.html

GPATH EXAMPLE

REST-assured


```
body("places[0].'place name'",
equalTo("Beverly Hills"));
```

SERIALIZATION OF POJOS

REST Assured is able to convert POJO instances directly to XML or JSON (and back)

Useful when dealing with test data objects

Requires additional libraries on the classpath

- Jackson or Gson for JSON
- JAXB for XML

AUTHENTICATION

REST assured also supports several authentication schemes, for example Basic, OAuth, digest, certificate etc.

There are two types of basic authentication -preemptive and challenged.

Preemptive Basic Authentication: This will send the basic authentication credential even before the server gives an unauthorized response in certain situations, thus reducing the overhead of making an additional connection.

given().auth().preemptive().basic("username", "password").when().get("/secured/hello").then().statusCode(200);

Challenged Basic Authentication : When using "challenged basic authentication" REST Assured will not supply the credentials unless the server has explicitly asked for it. This means that REST Assured will make an additional request to the server

given().auth().basic("username", "password").when().get("/secured/hello").then().statusCode(200)

Oauth2 Authentication

given().auth().oauth2("myAuthenticationToken").when().get("https://my.very.secure/api").then().assertThat().statusCode(200);

EXAMPLE: SERIALIZATION

```java
public class Address {

    private String street;
    private int houseNumber;
    private int zipCode;
    private String city;

    public Address(String street, int houseNumber, int zipCode, Str

        this.street = street;
        this.houseNumber = houseNumber;
        this.zipCode = zipCode;
        this.city = city;
    }

    public String getStreet() { return this.street; }

    public int getHouseNumber() { return this.houseNumber; }
```

POJO representing an address

```java
@Test
public void serializeAddressToJson() {

    Address myAddress = new Address( street: "My street",  houseNumber: 1,  zipCode: 1234,  city: "Amsterdam");

    given().
        body(myAddress).
    when().
        post( s: "http://localhost:9876/address").
    then().
        assertThat().
        statusCode(200);
}
```

```
Body:
{"street":"My street","houseNumber":1,"zipCode":1234,"city":"Amsterdam"}
```

EXAMPLE: SERIALIZATION

Instantiating it in a test and sending it as a request body for a POST method:

EXAMPLE: DESERIALIZATION

```
@Test
public void deserializeJsonToAddress() {

    Address myAddress =

        given().
        when().
            get( s: "http://localhost:9876/addr
            as(Address.class);

    Assert.assertEquals( expected: "Amsterdam", m
```

We can also convert a JSON (or XML) body back to an instance of a POJO

After that, we can do some verifications on it:

PROBLEM STATEMENT-1

Get:

From url: https://jsonplaceholder.typicode.com/comments
Retrieved the last entry from the response and print the email on console


```
{
  "postId": 100,
  "id": 500,
  "name": "ex eaque eum natus",
  "email": "Emma@joanny.ca",
  "body": "perspiciatis quis doloremque\nveniam nisi eos velit sed\nid totam inventore voluptatem laborum et eveniet\naut aut aut maxime quia temporibus ut omnis"
}
```

PROBLEM STATEMENT-2

Post: Post the following request to the URL given below

Endpoint : http://localhost:3000/comments

Post a request:{

“Id”: random number,

“Name”: ”Sumit Ghosh”

“Designation”: “TL”

}

Q&A

IDEMPOTENT

Wikipedia definition

Idempotence is the property of certain operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application.

Common HTTP Methods

CenturyLink®