API

Class 2

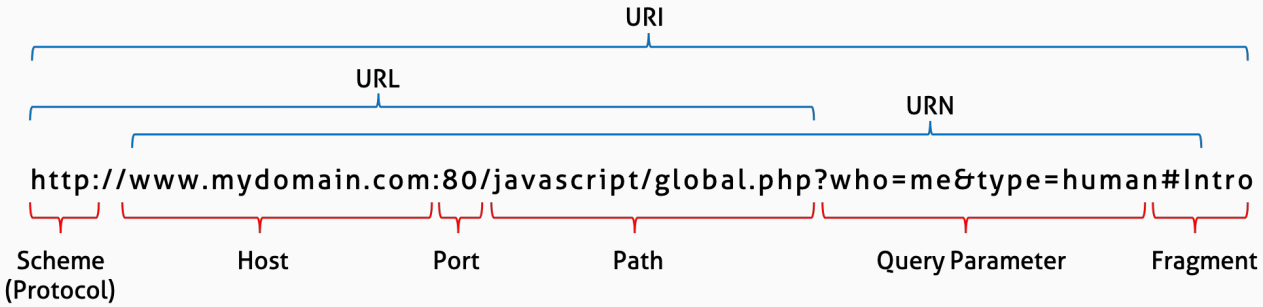
Agenda

What is RestAssured?
Types of Parameters

URL vs URI

URL - Uniform Resource Locator (where)

URI - Uniform Resource Identifier



API Endpoint

An endpoint is one end of a communication channel.
When an API interacts with another system, the touchpoints of this communication are considered endpoints.
An endpoint is simply a unique URL that represents an object or collection of objects.
Each endpoint is the location from which APIs can access the resources they need to carry out their function.

RestAssured

REST Assured is a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs.

REST Assured supports BDD syntax

REST Assured can be used easily in combination with existing unit testing frameworks, such as JUnit and TestNG.

RestAssured

REST Assured BDD Syntax:

given() → prepare request
when() → send the request
then() → validate request

```
given().
      when().
      get("https://got-quotes.herokuapp.com/quotes?
      char=tyrion").prettyPrint();
```

GET Request

Retrieves information.

GET requests must be safe and idempotent, meaning regardless of how many times it repeats with the same parameters, the results are the same.

Example1: **Get/api/getAllStudentProfiles**
View a profile of available syntax student

Example2: **Get/api/getStudentProfile/{studentId}**
View a profile of specific syntax student

GET Request

This HTTP method is used to **read/retrieve** resource representation only

Status Codes:
200 (OK) –> If GET API finds the requested resource.
404(Not Found) –> If GET API does not find the requested resource.
400 (Bad Request) –> If GET request is not formed properly.

Parameters

Parameters are options you can pass with the endpoint.

There are two types of parameters:

- **path parameters**
- **query string parameters**

Not all endpoints contain each type of parameter.

Parameters

Path parameters: Parameters within the path of the endpoint, before the query string (?). These are usually set off within curly braces.

/api/deleteStudentProfile/{studentId}

Query string parameters: Parameters in the query string of the endpoint, after the ?.

https://got-quotes.herokuapp.com/quotes?char=tyrion

Parameters

```
@Test
public void pathParametersTest() {

    RestAssured.baseURI = "http://pure-ravine-92491.herokuapp.com/syntax";

    given().
        pathParam("studentId", 80).
    when().
        get("/api/getStudentProfile/{studentId}").
        prettyPrint();
}

@Test
public void queryParametersTest() {

    RestAssured.baseURI = "https://got-quotes.herokuapp.com";
    given().
        queryParam("char", "tyrion").
    when().
        get("/quotes").
        prettyPrint();
}
```

Hamcrest Matcher

Hamcrest is a library we use to perform assertions.

Hamcrest allows checking for conditions in our code using existing matchers classes.

To use Hamcrest matchers in JUnit we use the **assertThat** statement followed by one or several matchers.

```
import static org.hamcrest.Matchers.*;

public class Assertions {

    @Test
    public void getWithAssertion1() {

        RestAssured.baseURI="http://pure-ravine-92491.herokuapp.com/syntax";

        given().
            pathParam("studentId", "81").
        when().
            get("/api/getStudentProfile/{studentId}").
        then().assertThat().
            body("firstName", equalTo("Sandesh"));
    }
```

Hamcrest Matcher

`allOf` - matches if all matchers match (short circuits)

`anyOf` - matches if any matchers match (short circuits)

`not` - matches if the wrapped matcher doesn't match and vice

`equalTo` - test object equality using the equals method

`is` - decorator for equalTo to improve readability

`hasToString` - test Object.toString

`instanceOf`, `isCompatibleType` - test type

`notNullValue`, `nullValue` - test for null

`sameInstance` - test object identity

`hasEntry`, `hasKey`, `hasValue` - test a map contains an entry, key or value

`hasItem`, `hasItems` - test a collection contains elements

`hasItemInArray` - test an array contains an element

`closeTo` - test floating point values are close to a given value

`greaterThan`, `greaterThanOrEqualTo`, `lessThan`, `lessThanOrEqualTo`

`equalToIgnoringCase` - test string equality ignoring case

`equalToIgnoringWhiteSpace` - test string equality ignoring differences in runs of whitespace

`containsString`, `endsWith`, `startsWith` - test string matching

POST Request

HTTP POST requests are used to create new resources. The basic idea is to pull data out of the HTTP request body and use it to create a new row in the database.

Request that the resource at the URI do something with the provided entity.

Often POST is used to create a new entity, but it can also be used to update an entity.

A HTTP POST method is used to create a new resource in collection of resources with a request body passed as a JSON/XML.

POST Request

Example: **POST /api/createStudentProfile**
-create syntax student profile

POST is not safe method as it is related to data creation.
It is also not idempotent and invoking two identical POST requests will result in two different resources containing the same information with just different resource ids.

Status Codes:
201(Created)–> If resource is created successfully at the end point,

PUT Request

PUT requests will be used to make updates to existing resources. It's important to note that PUT is used to replace the entire resource – it doesn't do partial updates.
PUT can create a new entity or update an existing one.
PUT is not a safe method as it performs data creation and modifications

A PUT request is idempotent. Idempotency is the main difference between the expectations of PUT versus a POST request.

Example: **PUT /api/updateStudentProfile**

DELETE Request

It is used to **delete** a resource identified by a URI. In other words, a 204 status with no body, or the JSEND-style response and HTTP status 200 are the recommended responses.
Request that a resource be removed; however, the resource does not have to be removed immediately. It could be an asynchronous or long-running request.

Example: DELETE
/api/deleteStudentProfile/{studentId}

- delete available syntax student profile

DELETE Request

It returns 200 (OK) ,204 (No Content) status code.

It may return as 202 (Accepted) status code if request is queued.

It is not a safe method as it performs on modification of data.

If we hit the same request again after first hit, it will give you 404 (Not Found) .
So DELETE request are idempotent after second call onward.

HTTP Response Codes

200 OK: - Code indicates that the request was successful.
201 Created:- Code indicates that request was successful and a resource was created. It is used to confirm success of a PUT or POST request.
400 Bad Request:- It happens especially with POST and PUT requests, when the data does not pass validation, or is in the wrong format.
404 Not Found:- response indicates that the required resource could not be found.
401 Unauthorized:- error indicates that you need to perform authentication before accessing the resource.
405 Method Not Allowed:- HTTP method used is not supported for this resource.
409 Conflict:- Conflict request to create the same resource twice.
500 Internal Server Error:- Occurs due to some error on Server side.