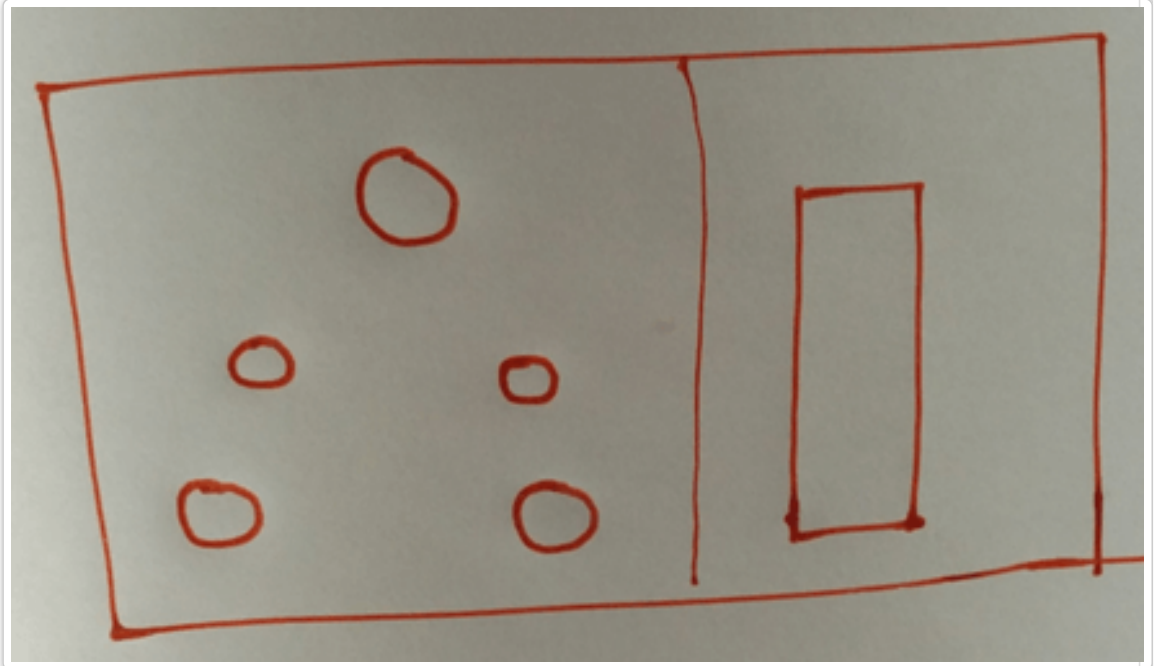


API Testing

API Testing

I have been trying to explain this api in simple way but somehow I was having difficulty but finally got an explanation in simple terms, so if something is wrong with explanation please do forgive me.



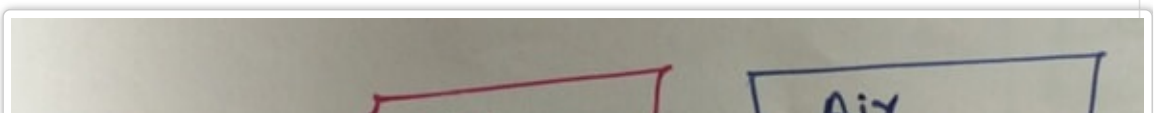
I hope You have seen this thing in your everyday life, if not (at least you must be sure that this is not a lemon).

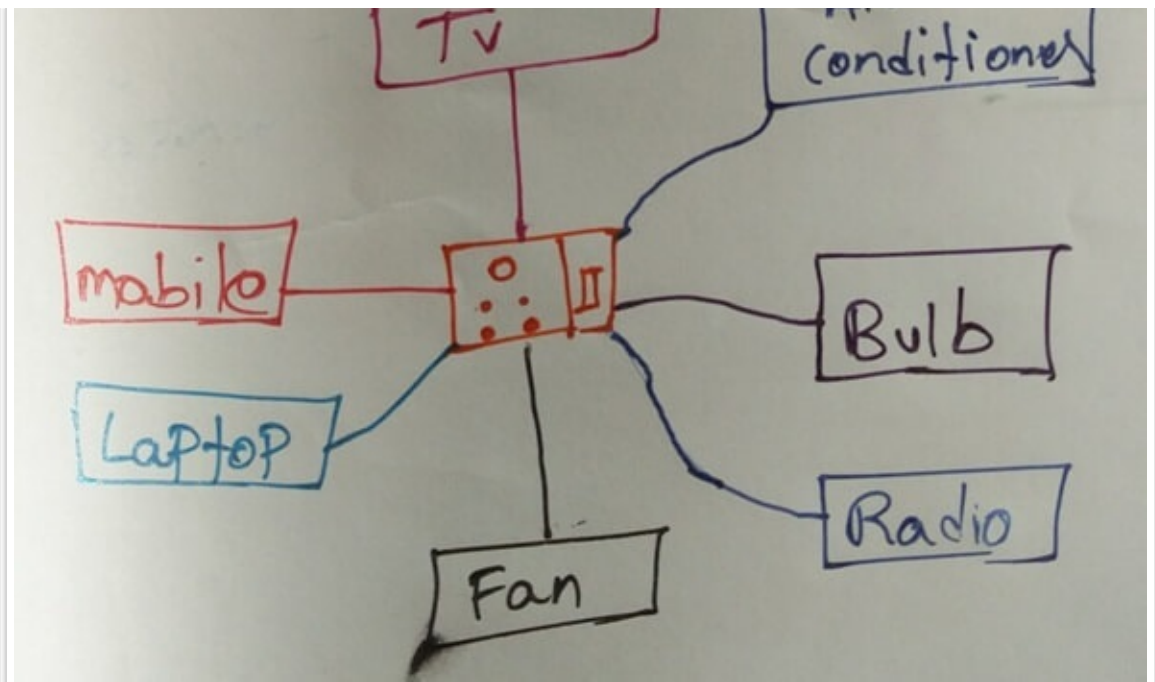
This is an Indian electrical socket, if you are from another county then instead of circle rounds then you might have come across vertical ones.

certainly I am not electrical engineer but Mechanical engineer

An api is nothing but similar to electrical socket

This socket provides 240V power supply, we can connect electrical devices like fan, TV, Washing Machine, Radio, Air Conditioner and so on.

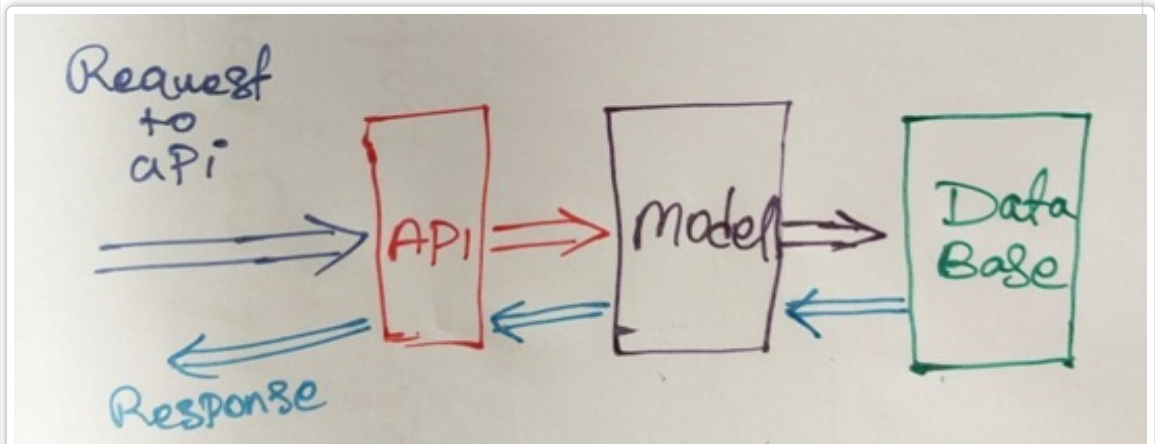




Similar to this socket is api exposes an URL, to which different websites can connect and those website could be developed using different technologies. Major use of the api is exposing and modifying the data.

Fan, Radio, TV are developed for different purposes but they will connect to same Socket and they will achieve their purpose either by adjusting voltage.

Similarly, using api the developers will build their application and UI based on their website need and manipulate it.



Can you guess, Which take less time to execute UI or API ?

You are right, The one which doesn't have UI less i.e API.

It doesn't mean every website will have API, based on the functionalities developers use APIs, usage of API is solely dependent on the business.

How API testing Works

API testing is not a big deal, as everyone say you have to test the URL and check the Response body and response status code.

Is it All about API Testing ? Nah, testing an URL is a first part of API Testing. API testing couple of stages (I am not sure number of stages that is why mentioned 'couple').

API is tested in multiple stages but all these stages are solely dependent on your team and organization.

Hit the API url for the given data (unique id or something)

Check the database for the same data with that unique id

Navigate to the UI of the API and verify the details of the product

Hit the API URL with required Operation, Operations could be Post, Put, Delete

Hit the GET url of the API and verify the changes

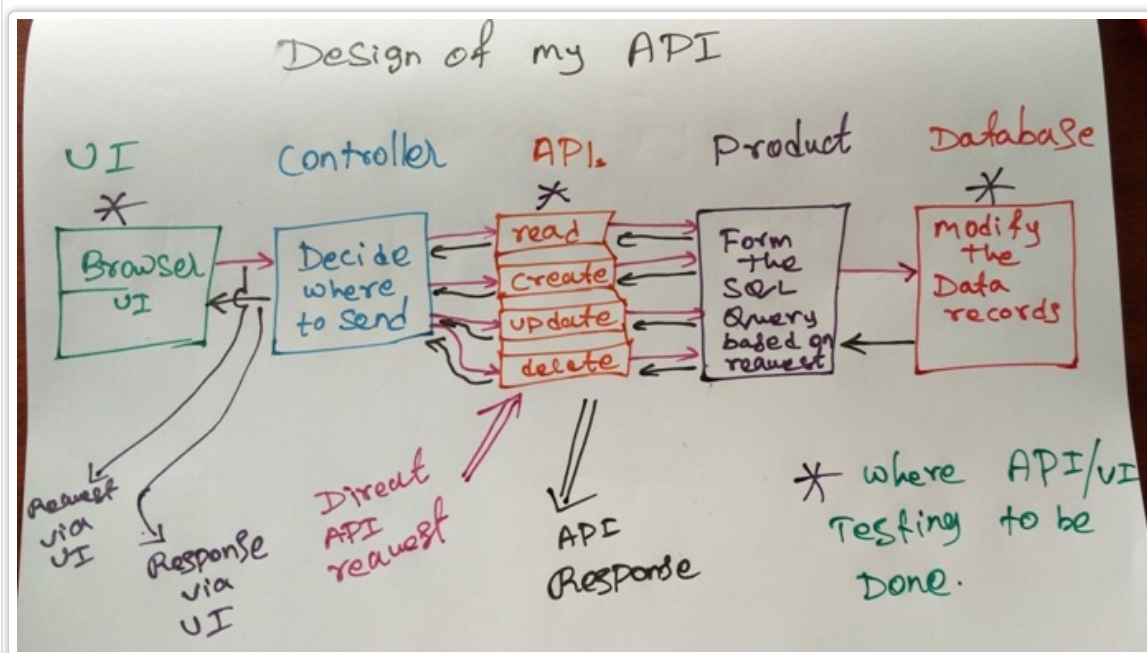
Test the database whether API operation is successful, and data is changed or removed

Navigate to the UI of the API and verify the product details

We would be performing individual operations and once we learn we will learn API Testing processes, then we would be doing all above mentioned scenario

Our API URL

I don't want my reader to depend on the third-party APIs, because of which I have created a dummy API (I am not developer so please do expect some issues and error).



I have created this api

using the tutorial and example present in the Internet, this API has following features.

This API is built using PHP and angular with reach

You can create products using this API

You can delete the created Product or exiting Products

You can edit new/Existing products

This API it not behind authentication

You can view all the products or a specific products bases on id, name, price but you cannot mix them.

You are not restricted to perform any operation on this api and it doesn't restrict sharing or performing changes

This API has an UI at : <https://chercher.tech/sample/api-ui>

Apart from these thanks to gitHub, and I want you to perform or test the functionalities of the API UI using above URL so that you will be familiar with operations

The UI and the APi both point to the same place in Db and the only change is how these call the target.

It is something like we can call a person with first name as well as with second name, if no other person is with same name. But you cannot do the same thing with my name because 3 out of 10 people have my name. silly.

Below are few End points of our api, these end point change from product to product and sometimes some products will have only one end point for all the urls

Sample API for GET all : <https://chercher.tech/sample/api/product/read>

Sample API for GET Specific: <https://chercher.tech/sample/api/product/read?id=90>

Sample API for PUT : <https://chercher.tech/sample/api/product/create>

Sample API for POST : <https://chercher.tech/sample/api/product/update>

Sample API for DELETE : <https://chercher.tech/sample/api/product/delete>

Manipulate the API using just URL bar :

Sample API for GET all : <https://chercher.tech/sample/api/product/read>

Sample API for GET Specific: <https://chercher.tech/sample/api/product/read?id=50> instead of **?id** you ca also use name or price wit respective value, you must not use any single/Double quotes in url even for string parameters as well.

Sample API for PUT : <https://chercher.tech/sample/api/product/create?name=xyz&description=desc of xyz&price=30>

Sample API for POST : <https://chercher.tech/sample/api/product/update?id=40&name=xyz&description=desc of xyz&price=30>

Sample API for DELETE : <https://chercher.tech/sample/api/product/delete?id=50>

Why API testing

API testing is done to know at what point code is not giving output. API testing is very essential. API testing helps to find many kind of bugs such as

Fails to handle error condition

Missing functionality

Duplicate functionality

Difficulty in connecting and getting response from API.

Request and response time(Performance Issue)

Security Issues

Threading issues

Improper errors/warning to caller

JSON or XML, responses

API Automation with Rest Assured

Now a days APIs are playing an ever more important role in software trends (because we can use same APi for mobile applications, the Internet of Things, etc.), proper automated testing of these APIs is becoming important.

There are many different tools out there that can assist you in writing these automated tests at the API level.

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

Integrate RestAPI with Selenium :

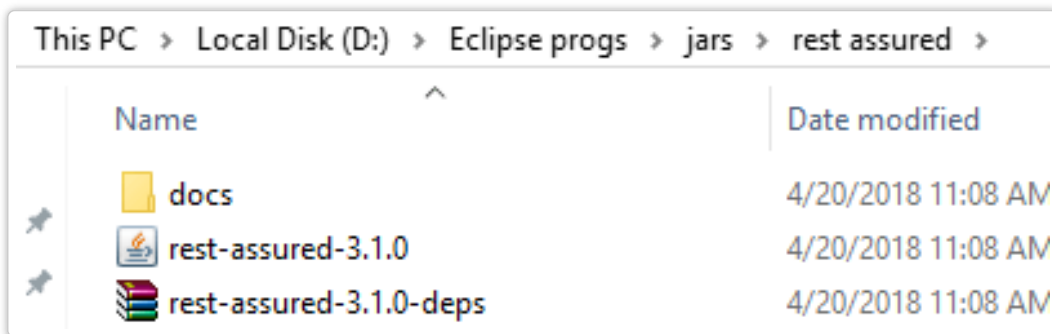
Navigate to Rest Assured URL : <https://github.com/rest-assured/rest-assured/wiki/Downloads>

Click on the rest-assured-version link, and it should start downloading the zip file

Current direct downloads

File	Description
rest-assured-3.1.0-dist.zip	REST Assured 3.1.0 (including source and javadocs and dependencies, XPath and JsonPath)
json-path-3.1.0-dist.zip	JsonPath 3.1.0 (standalone, including source and javadocs and dependencies)

Unzip the folder and You should be able to see below the things inside

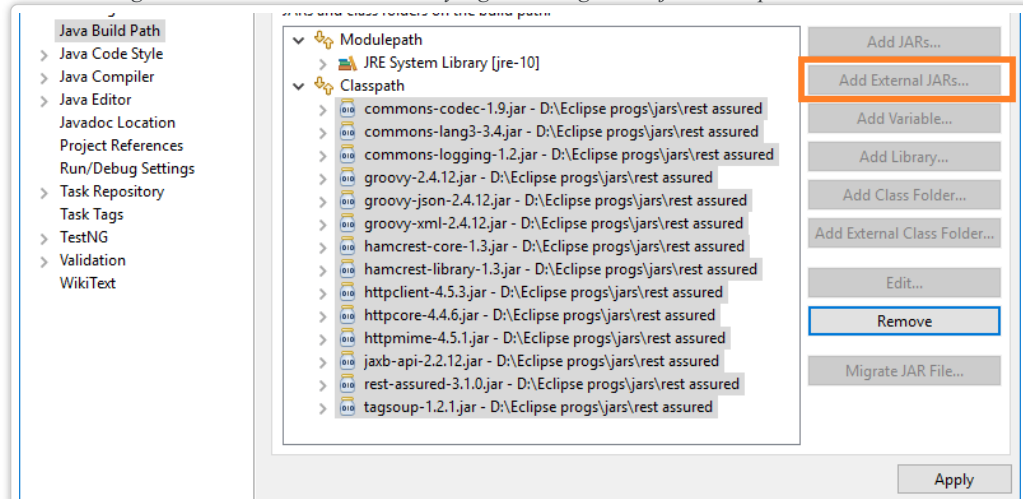


Unzip the rest-assured-version-deps.zip file

Place the jar file under required folder in your framework

If you have project, then add these jars along with the jars present in the docs folder using Add External Jar option on Eclipse

You can navigate to Add External Jar Place by right clicking on Project > Properties > Java Build Path



Methods present in REST API (important) :

There are mainly 4 methods involve in API Testing, those are :

GET

POST

DELETE

PUT

GET method in Rest API

The GET method is used to extract information from the given server using a given URI. While using GET request, it should only extract data and should have no other effect on the data.

Status Code of GET


200 OK: The request was successful, and the response body contains the representation requested.

302 FOUND: A common redirect response; you can GET the representation at the URI in the Location response header.

304 NOT MODIFIED: There is no new data to return.

get method is like a SELECT query in SQL which displays records present on the database

Below image is actual screenshot of api



```
// read products
function readAll() { // select all query
    $query = "SELECT
                id, name, description, price, created
            FROM
                " . $this->table_name . "
            ORDER BY
                id DESC";

    // prepare query statement
    $stmt = $this->conn->prepare( $query );
    // execute query
    $stmt->execute();
    return $stmt;
}
```

With simple steps, we

can request the api, which request the server to provide the details from the database.

End Point : <https://chercher.tech/sample/api/product/read>

Above url read all the data present in the database, but if add any specific id along with we will get only that particular data in our Rest API.

Below URl will only return the details for the id 90 :

End Point : <https://chercher.tech/sample/api/product/read?id=90>

Steps for the GET using RestAPI

Create a @Test method with TestNG in Java Project

Request the server using the given end point using get() method from the RestAssured class

```
// request the server
Response response = RestAssured.get("https://chercher.tech/sample/api/product/read");
```


Get body from the response of the using `getBody()` from the response object

```
// store the response body in string
String responseBody = response.getBody().asString();
```

Print the Response body

Get the status code for the response using `getStatusCode()` method

```
System.out.println(response.getTimeIn(TimeUnit.MILLISECONDS));
```

```
import java.util.concurrent.TimeUnit;
import org.testng.annotations.Test;
import io.restassured.RestAssured;
import io.restassured.response.Response;
public class FirstAPITest {
    @Test
    public void readAllDetails()
    {
        // request the server
        Response response = RestAssured.get("https://chercher.tech/sample/api/product/read");
        // store the response body in string
        String responseBody = response.getBody().asString();
        // print the response
        System.out.println("Response Body is => " + responseBody);
        // store the response code
        int responseStatusCode = response.getStatusCode();
        System.out.println("*****");
        System.out.println("Status Code => " + responseStatusCode);
        System.out.println(response.getTimeIn(TimeUnit.MILLISECONDS));
    }
}
```

Output of the rest API `get()` method

```
{
  "id": "2",
  "name": "Google Nexus 4",
  "description": "The most awesome phone of 2013!",
  "price": "299",
  "created": "2014-06-01 01:12:26"
}
]
}
*****
Status Code => 200
1400
```

As I said earlier you can

get the single data as well with api, by adding a parameter.


```
import java.util.concurrent.TimeUnit;
import org.testng.annotations.Test;
import io.restassured.RestAssured;
import io.restassured.response.Response;
public class FirstAPITest {
    @Test
    public void readAllDetails()
    {
        // request the server
        Response response = RestAssured.get("https://chercher.tech/sample/api/product/read?id=90");
        // store the response body in string
        String responseBody = response.getBody().asString();
        // print the response
        System.out.println("Response Body is => " + responseBody);
    }
}
```

PUT method in API testing

Put method is different from get() method, put method creates details or resource the server/database.

It is upto the developer whether a end point API url support the both Create an Update or either of them, the API we are using in below example will create the resource but will not have capability to update

When the target resource exists it overwrites that resource with a complete new body. That is PUT method is used to CREATE or UPDATE a resource.

Status Codes of PUT

201 OK: The request was successful, we updated the resource and the response body contains the representation.

202 ACCEPTED: The request has been accepted for further processing, which will be completed sometime later.

put method is like a UPDATE query in SQL which inserts or updates a record depending upon whether the given record exists

```
// update the product
function update(){
    // update query
    $query = "UPDATE
            " . $this->table_name . "
            SET
            name = :name,
            price = :price,
            description = :description
            WHERE
```

```
id = :id";  
  
// prepare query statement  
$stmt = $this->conn->prepare($query);
```

put() method in API may return 201 status code with message 'Created' if everything is successful.

We have to prepare the statement for the request using given() method present in RestAssured

```
RequestSpecification reqSpec = RestAssured.given();
```

We can directly pass the values or we can use JSON object to pass the details that we want to create or update

```
JSONObject jo = new JSONObject();  
jo.put("name", "myname");  
jo.put("description", "that is my name");  
jo.put("price", "122222");
```

Register the data into the request URL/API

```
reqSpec.body(jo.toString());
```

Use put() method from the request to put the data into database

```
Response resp = reqSpec.put("https://chercher.tech/sample/api/product/create");
```

statusCode() method will fetch the status code of the request, if successful then it will return 201

Complete code for Putting a details into database

```
@Test  
public void putDetails()  
{  
    RequestSpecification reqSpec = RestAssured.given();  
  
    JSONObject jo = new JSONObject();  
    jo.put("name", "myname");  
    jo.put("description", "that is my name");  
    jo.put("price", "122222");  
    reqSpec.body(jo.toString());  
  
    Response resp = reqSpec.put("https://chercher.tech/sample/api/product/create");  
  
    System.out.println("Response code => " + resp.statusCode());  
}
```

```
[RemoteTestNG] detected TestNG version 6.14.2  
WARNING: An illegal reflective access operation has occurred  
WARNING: Illegal reflective access by org.codehaus.groovy.reflection.Cache
```

```

WARNING: Please consider reporting this to the maintainers of org.codehaus
WARNING: Use --illegal-access=warn to enable warnings of further illegal r
WARNING: All illegal access operations will be denied in a future release
Response code => 201
Response code => Product was created
PASSED: putDetails

```

You might want to

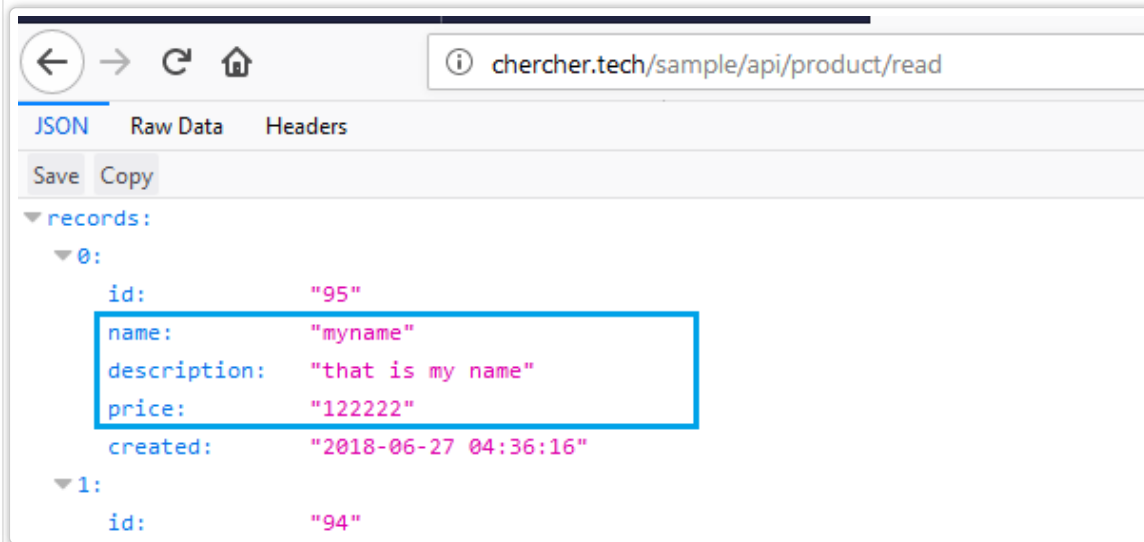
```

=====
Default test
Tests run: 1, Failures: 0, Skips: 0

```

check the api to see whether it got reflected or not, Please visit below link, there could be difference in id when you run same code.

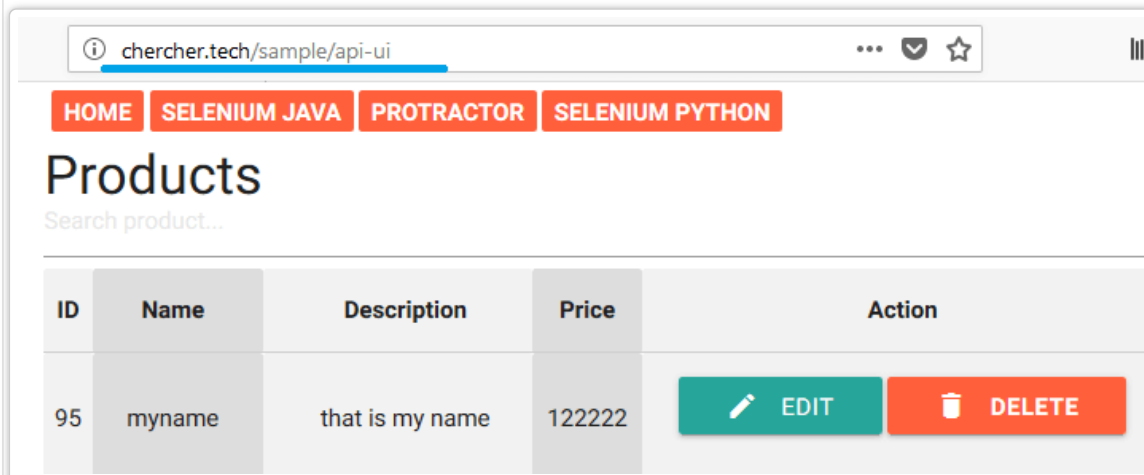
<https://chercher.tech/sample/api/product/read>



If you wish to see the

change in the UI then please do visit the UI of Our Api, which resides at :

<https://chercher.tech/sample/api-ui>



POST Method in Rest API

A POST request is used to create new entity or modify existing one. It can also be used to send data to the server, for example, Product name, customer information, file upload, etc. using HTML forms.

"Post" means "after"; if you have a collection of entities and you tack a new one onto its end, you have posted to the collection.

You can't post an existing entity, and it's common (though not always required) to use the collection's URI to post.

Status Codes of POST

201 OK: The request was successful, we updated the resource and the response body contains the representation.

202 ACCEPTED: The request has been accepted for further processing, which will be completed sometime later.

post method is like a INSERT query in SQL which always creates a new record in database.

```
// create product
function create(){

    // query to insert record
    $query = "INSERT INTO
              " . $this->table_name . "
              SET
              name=:name, price=:price, description=:description,
              created=:created";

    // prepare query
    $stmt = $this->conn->prepare($query);
```

Every API will

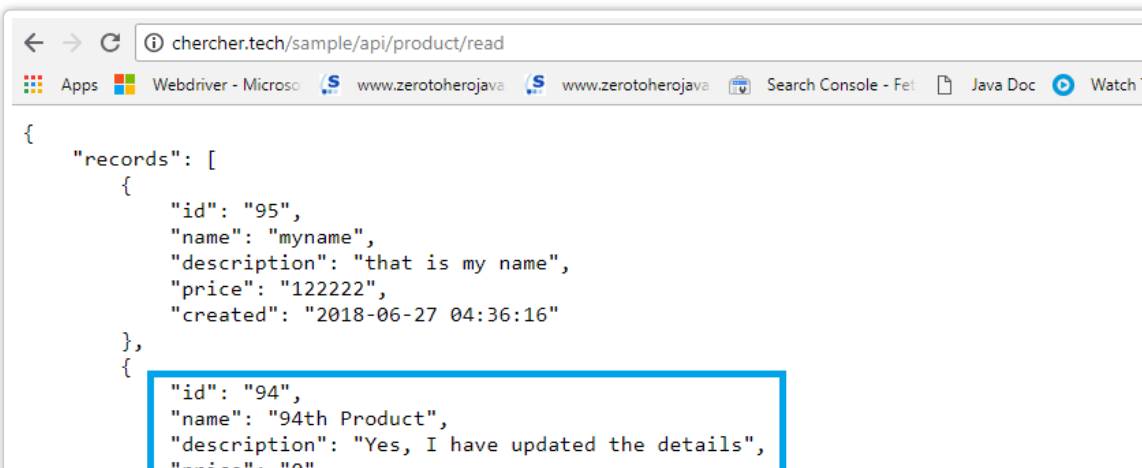
provide some uniquely parameter for every products, using that parameter we have to update the details for this example I would be using **id**

```
@Test
public void postDetails()
{
    // request the server
    RequestSpecification reqSpec = RestAssured.given();

    JSONObject jo = new JSONObject();
    jo.put("id", "94");
    jo.put("name", "94th Product");
    jo.put("description", "Yes, I have updated the details");
    jo.put("price", "0.0");
    reqSpec.body(jo.toString());

    Response resp = reqSpec.post("https://chercher.tech/sample/api/product/update");
}
```

Verify whether details got updated or not, using API <https://chercher.tech/sample/api/product/read>



```
{
  "records": [
    {
      "id": "95",
      "name": "myname",
      "description": "that is my name",
      "price": "122222",
      "created": "2018-06-27 04:36:16"
    },
    {
      "id": "94",
      "name": "94th Product",
      "description": "Yes, I have updated the details",
      "price": "0.0"
    }
  ]
}
```

You might want to

```
    "price": 0,
    "created": "2018-06-27 04:34:51"
  },
```

verify the Ui of the Api., Visit :

<https://chercher.tech/sample/api-ui>

Products

Search product...

ID	Name	Description	Price	Action
95	myname	that is my name	122222	EDIT DELETE
94	94th Product	Yes, I have updated the details	0	EDIT DELETE

Delete method in Rest API

DELETE- Removes data from the target resource/ database given by a URI.

Status Code of DELETE

202 ACCEPTED: The request has been accepted for further processing, which will be completed sometime later.

204 OK: The request was successful; the resource was deleted.

put method is like a UPDATE query in SQL which inserts or updates a record depending upon whether the given record exists

```
// delete the product
function delete(){

    // delete query
    $query = "DELETE FROM " . $this->table_name . " WHERE id = ?";

    // prepare query
    $stmt = $this->conn->prepare($query);
```

```
@Test
public void deleteDetails()
{
    // request the server
    RequestSpecification reqSpec = RestAssured.given();

    JSONObject jo = new JSONObject();
    jo.put("id", "94");

    reqSpec.body(jo.toString());

    Response resp = reqSpec.delete("https://chercher.tech/sample/api/product/delete");
```

Verify whether details got updated or not, using API :<https://chercher.tech/sample/api/product/read>

```
"records": [  
  {  
    "id": "95",  
    "name": "myname",  
    "description": "that is my name",  
    "price": "122222",  
    "created": "2018-06-27 04:36:16"  
  },  
  {  
    "id": "93",  
    "name": "dummy",  
    "description": "dummy description",  
    "price": "120",  
    "created": "2018-06-27 04:32:40"  
  },  
]
```

94 got deleted



You might want to

verify the Ui of the Api, Visit : <https://chercher.tech/sample/api-ui>

Products

Search product...

ID	Name	Description	Price	Action
95	myname	that is my name	122222	 EDIT  DELETE
93	dummy	dummy description	120	 EDIT  DELETE

PATCH

PATCH is used to partially update an existing entity with new information. You can't patch an entity that doesn't exist. You would use this when you have a simple update to perform, e.g. changing a user's name.

patch method is like a UPDATE query in SQL which sets or updates selected columns only and not the whole row.

Complete API Testing

As I explained earlier API testing mean not just testing the Url but testing the total process, so far we have learned how to perform individual units. Now combine all your testing skill and then perform below scenario.

Scenario : Create a Product and update price of the created product and delete the created product

Steps for the scenario :

Use the API to read and get the latest ID of the product

Open the UI of the API and check the id of the latest product

Open database connection and check whether the products is present (ignore this step as I might need your IP address of the system to provide access)

Create the Product with given details : name=radio, description=invented by Markoni, price=1200

Read the product details from the API using read end point

Open the UI of the api and verify the product details and id should match with API details of the above step

Change the price of the product using retrieved id, new price should be updated to 890 using API

Get the details of the specific product using API and verify the updated prices

Open the UI of the API and verify the product details

*On the UI using Edit Button Update the description to **this device plays songs***

Open the specific product using the API and verify the description

Delete the product using API and verify it is removed using API

Open UI of The APi and verify product is not present

Best Practices in API testing

First & foremost, test for the functionality – the basic request-response is working consistently.

Group Test cases by Test category like sanity, regression

For complete Test coverage, create test cases for all possible API input combinations.

Test cases should include both positive and negative scenarios. Negative test are for to see, how it handles unforeseen problems and loads making sure the API fails gracefully.

Add stress to the system through a series of API load tests.

Parameters selection should be explicitly mentioned in the Test case itself.

Prioritize API function calls so that it will be easy for testers to test in a timely fashion.

Automate API documentation creation with a standard like Swagger, but then run through the tests, making sure the documentation makes sense for all levels of user experience.

While writing test cases, make sure that no hard coded values are being used for test data. Test data should be written in separate text/csv file.

There should be reporting functionality that defines the output in user readable format.

All HTTP Status Codes

General

1xx Status codes

100 Continue : The HTTP 100 Continue informational status response code indicates that everything so far is OK and that the client should continue with the request or ignore it if it is already finished.

101 Switching Protocols : The HTTP 101 Switching Protocols response code indicates the protocol the server is switching to as requested by a client which sent the message including the Upgrade request header

102 Processing : An interim response used to inform the client that the server has accepted the complete request, but has not yet completed it.

2xx Status codes in API

Browsers generate a success indicator for the 2XX status code. So 2XX status codes should be used specify a successful request.

200 OK : The HTTP 200 OK success status response code indicates that the request has succeeded.

201 Created : Used for POST request to create resource

202 Accepted : Request accepted by server, but cannot respond immediately.

203 Non-Authoritative Information : Retrieve information expected from 202 request.

204 No Content : Response doesn't have a payload.

205 Reset Content : The HTTP 205 Reset Content response status tells the client to reset the document view, so for example to clear the content of a form, reset a canvas state, or to refresh the UI.

206 Partial Content : The server is successfully fulfilling a range request for the target resource by transferring one or more parts of the selected representation that correspond to the satisfiable ranges found in the request's Range header field

207 Multi-Status : A Multi-Status response conveys information about multiple resources in situations where multiple status codes might be appropriate.

208 Already Reported : The members of a DAV binding have already been enumerated in a previous reply to this request, and are not being included again.

226 IM Used : The server has fulfilled a GET request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.

3xx Status codes in API

The REST API developer should maintain old resources, in case he is migrating to new ones.

3XX series codes are displayed as errors in browser console and should be used to indicate resource relocation.

300 Multiple Choices : The HTTP 300 Multiple Choices redirect status response code indicates that the request has more than one possible responses. The user-agent or the user should choose one of them. As there is no standardized way of choosing one of the responses, this response code is very rarely used.

301 Moved Permanently : Server changed the URI and asking the client to use a new URI.

302 Found : Server wants to retain old URI, providing an alternate URI. Since Cool URIs don't change

303 See Other : The HyperText Transfer Protocol (HTTP) 303 See Other redirect status response code indicates that the redirects don't link to the newly uploaded resources but to another page, like a confirmation page or an upload progress page. This response code is usually sent back as a result of PUT or POST. The method used to display this redirected page is always GET.

304 Not Modified : Server instructing the client to use its cached results. 304 is considered a redirect because the server is redirecting the client to its own cache for the response and not to another URI.

307 Temporary Redirect : The target resource resides temporarily under a different URI and the user agent MUST NOT change the request method if it performs an automatic redirection to that URI.

308 Permanent Redirect : The target resource has been assigned a new permanent URI and any future references to this resource ought to use one of the enclosed URIs.

4xx Status codes in API

When the client makes a mistake, the server should notify the client of 4XX error. The most popular being 404. When the server cannot find the resource, the client requested.

Browsers show errors in their console for 4XX series, even when they necessarily are errors. For example, when the resource is deleted, the server SHOULD return 410 instead of 200 stating that the resource has been deleted.

400 BAD REQUEST: The request was invalid or cannot be otherwise served. An accompanying error message will explain further. For security reasons, requests without authentication are considered invalid and will yield this response.

401 UNAUTHORIZED: The authentication credentials are missing, or if supplied are not valid or not sufficient to access the resource.

403 FORBIDDEN: The request has been refused. See the accompanying message for the specific reason (most likely for exceeding rate limit).

404 NOT FOUND: The URI requested is invalid or the resource requested does not exists.

405 Method Not Allowed : The HyperText Transfer Protocol (HTTP) 405 Method Not Allowed response status code indicates that the request method is known by the server but has been disabled and cannot be used.

406 NOT ACCEPTABLE: The request specified an invalid format.

407 Proxy Authentication Required : The HTTP 407 Proxy Authentication Required client error status response code indicates that the request has not been applied because it lacks valid authentication credentials for a proxy server that is between the browser and the server that can access the requested resource.

408 Request Timeout : The server did not receive a complete request message within the time that it was prepared to wait.

409 Conflict : The HTTP 409 Conflict response status code indicates a request conflict with current state of the server.

410 GONE: This resource is gone. Used to indicate that an API endpoint has been turned off.

411 Length Required : 411 Length Required client error response code indicates that the server refuses to accept the request without a defined Content-Length header.

412 Precondition Failed : One or more conditions given in the request header fields evaluated to false when tested on the server.

413 Payload Too Large : The server is refusing to process a request because the request payload is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request. If the condition is temporary, the server SHOULD generate a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

414 Request-URI Too Long : The HTTP 414 URI Too Long response status code indicates that the URI requested by the client is longer than the server is willing to interpret.

415 Unsupported Media Type : The HTTP 415 Unsupported Media Type client error response code indicates that the server refuses to accept the request because the payload format is in an unsupported format. The format problem might be due to the request's indicated Content-Type or Content-Encoding, or as a result of inspecting the data directly.

416 Requested Range Not Satisfiable : None of the ranges in the request's Range header field overlap the current extent of the selected resource or that the set of ranges requested has been rejected due to invalid ranges or an excessive request of small or overlapping ranges

417 Expectation Failed : The HTTP 417 Expectation Failed client error response code indicates that the expectation given in the request's Expect header could not be met.

418 I'm a teapot : The HTTP 418 I'm a teapot client error response code indicates that the server refuses to brew coffee because it is a teapot. This error is a reference of Hyper Text Coffee Pot Control Protocol which was an April Fools' joke in 1998.

421 Misdirected Request : The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.

422 Un-processable Entity : 422 Un-processable Entity response status code indicates that the server understands the content type of the request entity, and the syntax of the request entity is correct, but it was unable to process the contained instructions.

423 Locked : The source or destination resource of a method is locked.

424 Failed Dependency : The method could not be performed on the resource because the requested action depended on another action and that action failed.

426 Upgrade Required : The HTTP 426 Upgrade Required client error response code indicates that the server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol. The server sends an Upgrade header with this response to indicate the required protocol(s).

428 Precondition Required : The HTTP 428 Precondition Required response status code indicates that the server requires the request to be conditional. Typically, this means that a required precondition header, such as If-Match, is missing. When a precondition header is not matching the server side state, the response should be 412 Precondition Failed.

429 TOO MANY REQUESTS: Returned when a request cannot be served due to the application's rate limit having been exhausted for the resource.

431 Request Header Fields Too Large : 431 Request Header Fields Too Large response status code indicates that the server is unwilling to process the request because its header fields are too large. The request may be resubmitted after reducing the size of the request header fields. It can be used when the total number of request header fields is too large, or when a single header field is at too large.

444 Connection Closed Without Response : A non-standard status code used to instruct nginx to close the connection without sending a response to the client, most commonly used to deny malicious or malformed requests.

451 Unavailable For Legal Reasons : 451 Unavailable For Legal Reasons client error response code indicates that the user requested a resource that is not available due to legal reasons, such as a web page for which a legal action has been issued.

499 Client Closed Request : A non-standard status code introduced by nginx for the case when a client closes the connection while nginx is processing the request.

5xx Status codes in API

5xx errors are related to server, So server takes responsibility for these error status codes.

500 INTERNAL SERVER ERROR: Something is horribly wrong.

501 Not Implemented : 501 Not Implemented server error response code indicates that the request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

502 BAD GATEWAY: The service is down or being upgraded. Try again later.

503 SERVICE UNAVAILABLE: The service is up, but overloaded with requests. Try again later.

504 GATEWAY TIMEOUT: Servers are up, but the request couldn't be serviced due to some failure within our stack. Try again later.

505 HTTP Version Not Supported : 505 HTTP Version Not Supported response status code indicates that the HTTP version used in the request is not supported by the server.

506 Variant Also Negotiates : The server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper end point in the negotiation process.

507 Insufficient Storage : The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request. This condition is considered to be temporary.

508 Loop Detected : The server terminated an operation because it encountered an infinite loop while processing a request with "Depth: infinity". This status indicates that the entire operation failed.

510 Not Extended : The policy for accessing the resource has not been met in the request. The server should send back all the information necessary for the client to issue an extended request.

511 Network Authentication Required : The HTTP 511 Network Authentication Required response status code indicates that the client needs to authenticate to gain network access. This status is not generated by origin servers, but by intercepting proxies that control access to the network.

599 Network Connect Timeout Error : This status code is not specified in any RFCs, but is used by some HTTP proxies to signal a network connect timeout behind the proxy to a client in front of the proxy.

Recommended Readings

[Selenium interview questions](#)

[Selenium tricky interview questions](#)

[selenium framework interview questions](#)

[Core Java Interview Questions Set 1](#)

[Selenium Relative Locators But I am hesitant to use](#)

[Robot class in selenium](#)

Tags & Hooks in Cucumber with Selenium | BDD

Parameterize Cucumber BDD with selenium