## ######################### ➔API Testing ➔ #########################

## Q1: How do you perform API testing?

**ANS:** We perform API testing by using postman tool. We get the URI and JSON Payload from dev team and also get the Authentication details (User name and Pwd) from dev team then, we do modification in the payload based on our test cases & check the response accordingly. I do test for REST APIs

## Q2: Methods in API testing:

**Ans: Get Method:** When we want to fetch data from the server then we use GET method to fetch the data.

**POST Method:** When we want to create some new resource on the server then we use POST method.

**PUT Method:** When we want to update any existing resource then we use PUT method to modify. If parameter which we want to update is available then it modifies that parameter if that parameter is not available then it creates that parameter.

**PATCH Method:** When we want to do partially update then we use PATCH method to update. We pass only that parameter which we want to update, we don't have to pass complete payload to update the resource.

## Difference between POST, PUT & PATCH: Post method is used to create new user/entity on the server.

## First let us see that what each method does. Let suppose we have created a user through POST method like below:

```
{
  "id": 1,
  "name": "Sam Kwee",
  "email": "skwee357@olddomain.com",
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "10001"
}
```

The **PUT** included all of the parameters on this user in request Payload. Example below:

```
PUT /users/1
{
    "name": " Sam Kwee ",
    "email": "skwee357@gmail.com"       // new email address
    "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "10001"
```

}
but **PATCH** only included the one that was being modified (email).

```
PATCH /users/1
{
   "email": "himansu@gmail.com"      // new email address
}
```

When using **PUT**, it is assumed that you are sending the complete entity, and that complete entity ***replaces*** any existing entity at that URI. In the above example, the PUT and PATCH accomplish the same goal: they both change this user's email address. But PUT handles it by replacing the entire entity, while **PATCH** only updates the fields that were supplied, leaving the others alone. Patch do not replace the existing entity. It does modification in the existing entity. And in a resulted set it keeps the other fields along with modified field.

Updating through PATCH

```
PATCH /users/1
{"email": "saket@newdomain.com"}
```

Now get the result for user 1 through GET method to check that whether modification has done successful or not.

```
{
  "id": 1,
  "name": "Sam Kwee",
  "email": "saket@newdomain.com",    // the email changed, yay!
  "address": "123 Mockingbird Lane",
  "city": "New York",
  "state": "NY",
  "zip": "10001"
}
```

Since PUT requests include the entire entity, if you issue the same request repeatedly, it should always have the same outcome (the data you sent is now the entire data of the entity). Therefore PUT is idempotent. i.e. It will not replace the existing entity in this case, if you issue the same request repeatedly.

**Using PUT wrong**

What happens if you use the above PATCH data in a PUT request?

```
PUT /users/1
{
   "email": "himansu@gmail.com"      // new email address
}
```

Now we are using PUT and supplying email only. Now this is the only thing (email) in this entity. Now it has resulted in data loss. That’s why we need to include all of the parameters while using PUT method.

GET /users/1
{
  "email": "skwee357@gmail.com"  // Now we have new email address only... and nothing else! The other parameters have gone. We loss the data.
}

Q: **When to use PUT & When to use Patch?**
Ans: Let's assume the below user details for an example:

```
{
 "id": 1,
 "name": "Sam Kwee",
 "email": "saket@newdomain.com",  // the email changed, yay!
 "address": "123 Mockingbird Lane",
 "city": "New York",
 "state": "NY",
 "zip": "10001"
}
```

Let's suppose you want to do some modification and you want to remove a field like 'address' Now you can include all of the parameters which you want as result set. So here you will update the new values for the fields which you want to update and we won't include 'address' in our new PUT request payload. Now you will get the result like below:

```
{
"id": 1,
 "name": "Sam Kwee",
 "email": "Ajay@newdomain.com",  // the email changed, yay!
 "city": "Washington", // the City changed
 "state": "NY",
 "zip": "10003"   // zip code changed
}
```

And we will use PATCH method when we want to modification in 'City' field.

```
PATCH /users/1
{"city": "Vegas"}
```

Now result set would be like below:

```
"id": 1,
 "name": "Sam Kwee",
 "email": "saket@newdomain.com",
 "address": "123 Mockingbird Lane",
 "city": "Vegas",    // City changed
 "state": "NY",
 "zip": "10001"
}
```

NOTE: So basically PUT is overriding/ Replacing the existing entity and PATCH is modifying the existing entity.

## Q3: Response Codes:

**1xx informational response –** the request was received, continuing process

**2xx successful –** the request was successfully received, understood, and accepted

**200 OK:** The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource.

**201 Created:** The request has been fulfilled, resulting in the creation of a new resource.

**202 Accepted:** The request has been accepted for processing, but the processing has not been completed.

**3xx redirection –** further action needs to be taken in order to complete the request

**4xx client error –** the request contains bad syntax or cannot be fulfilled**.**

**400 Bad Request:** The server cannot or will not process the request due to an apparent client error (e.g., malformed request syntax, size too large, invalid request message framing, or deceptive request routing).

**401 Unauthorized (RFC 7235):** Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided.

**403 Forbidden:** The request contained valid data and was understood by the server, but the server is refusing action. This may be due to the user not having the necessary permissions for a resource.

**404 Not Found:** The requested resource could not be found but may be available in the future.

**5xx server error –** the server failed to fulfil an apparently valid request.

**500 Internal Server Error**

A generic error message, given when an unexpected condition was encountered and no more specific message is suitable**.**

**501 Not Implemented**

The server either does not recognize the request method, or it lacks the ability to fulfil the request. Usually this implies future availability (e.g., a new feature of a web-service API).

**502 Bad Gateway**

The server was acting as a gateway or proxy and received an invalid response from the upstream server.

**503 Service Unavailable**

The server cannot handle the request (because it is overloaded or down for maintenance). Generally, this is a temporary state.

**504 Gateway Timeout**

The server was acting as a gateway or proxy and did not receive a timely response from the upstream server.

**Challenges while doing API testing:** There are following challenges in API testing:

- Initial Setup of **API Testing**. Manual **testing** helps confirm whether something works.
- Updating the Schema of **API Testing**.
- **Testing** Parameter Combinations.
- Sequencing the **API** Calls.
- Validating Parameters.
- Test Data management
- Integration challenges: APIs interact with each other through a set of defined rules known as contracts or protocols. Often these protocols are complicated and might be reason for hindrance to the proper integration and testing of the communication between components.

**What API information is exposed in Web Developer tools?**

Request headers, Response body, Response cookies.

**which type of encoding does postman accept authorization credentials?**

Postman accepts Base64 encoding only. Because it transmits the data into the textual form and sends it in easier form such as HTML form data.

**Can global scope variables have duplicate names in postman?**

Since global variables are global i.e. without any environment, global variables cannot have duplicate names. Local variables can have the same name but in different environments.

**What is a Postman Collection?**

A Postman Collection lets us group individual requests together. Simply it allows us to organize the requests into folders.

**What do you mean by postman monitors?**

The postman monitor is used for running collections. Collections are run till specified time defined by the user. Postman Monitor requires the user to be logged in. Monitor reports are shared by users over email on a daily/monthly basis.

**What do you understand by the term Postman Collection runners?**

A postman collection runner is used to perform Data-driven testing. The group of API requests are run in a collection for the multiple iterations with different sets of data.

**How do you remove local variables?**

Local variables are automatically removed once the tests have been executed.

**How can we stop executing requests or stop the collection run?**

postman.setNextRequest(null);

**How can we access a Postman variable?**

We can access a Postman variable by entering the variable name as {{var}}

**How can you iterate a request 100 times in Postman?**

By using Collection Runner

**What will execute first in a Collection Run?**

Pre-request scripts at the Collection level are executed first in a Collection run.

**How can we log requests and responses in Postman?**

We can view requests logs and response logs through the Postman Console window.

**What are the main challenges of API testing?**

The main challenges in API testing is

- Parameter Selection
- Parameter Combination
- Call sequencing

**What are the types of Bugs will API testing finds?**

The types of Bugs, API will find

- **Missing or duplicate functionality (Functionality Bugs):** Before looking at how well the API performs; the tests need to determine if the API functions work properly or not. This starts with checking basic functionally like creating and deleting data via API calls as appropriate. The tests also look for missing functionality bugs where the API calls for a feature that isn't there.
- Stress

**Reliability: Does API Testing Work Consistently?**

When we get the API to work once is a good start, but the API needs to work every time. API testing helps identify bugs if some integration is there across different modules.

## Security: **Does API Testing Protect Data Exchanges?**

Finally, API testing finds security-related bugs. API calls move data between endpoints, if it is unprotected, then could be abused by hackers. Even if the data isn't confidential. The API test examines if data being sent over HTTP is correctly encrypted.

- Unused flags
- Not implemented errors
- Inconsistent error handling
- Performance
- Multi-threading issues
- Improper errors

**Explain the API testing approach.**

**Answer: Mentioned below are the factors which determine the approach:**

- Write appropriate test cases for the APIs and use testing techniques like boundary value analysis, equivalence class, etc. for verifying the functionality.
- Verify the calls of the combination of two or more value-added parameters.
- Define the scope and basic functionality of the API program.
- Define the accurate input parameters.
- Test case execution and comparison of the results with expected results.
- Determining API behaviour under conditions like the connection with files, etc

**Difference between HTTP and HTTPs:** HTTP stands for Hypertext Transfer Protocol, and it is a protocol and syntax for presenting information – used for transferring data over a network. Most information that is sent over the Internet, including website content and API calls, uses the HTTP protocol.

The S in HTTPS stands for "secure." HTTPS uses TLS (or SSL) to encrypt HTTP requests and responses, so in the example above, instead of the text, an attacker would see a bunch of seemingly random characters.

"If a website uses HTTP instead of HTTPS, all requests and responses can be read by anyone who is monitoring the session. Essentially, a malicious actor can just read the text in the request or the response and know exactly what information someone is asking for, sending, or receiving."

**Difference between API and Web Services:** A **Web services** are any bit of services that makes it accessible over the Internet and normalizes its correspondence through XML encoding.

**API** stands for **Application Programming Interface**. It is a collection of communication conventions and subroutines. A developer can utilize different API apparatuses to make its program simpler and less complex. "All web services are APIs but not all the APIs are web services."

**Authentication and authorization** might sound similar, but they are more related to identity and access management (IAM). **Authentication** confirms that users are who they say they are, **means Authentication confirms the identity of the user and Authorization gives permission to those identified users to access a resource.**

In simple words Authentication means **“Who you are” It’s all about to proving correct identity.** And Authorization means **“What you can do” It’s all about access, means do you have permission to access something.**

**Authentication methods:** There are 4 most used authentication methods below**:**

**Basic:** HTTP Basic Authentication is rarely recommended due to its security vulnerabilities. This is the most straightforward method and the easiest. With this method, the sender places a username: password into the request header. The username and password are encoded with Base64, which is an encoding technique that converts the username and password into a set of 64 characters to ensure safe transmission. This method does not require cookies, session IDs, login pages, and other such specialty solutions, and because it uses the HTTP header itself.

**Bearer:** Bearer authentication (also called token authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens. The name “Bearer authentication” can be understood as “give access to the bearer of this token.” The bearer token allowing access to a certain resource or URL, usually generated by the server in response to a login request. The client must send this token in the Authorization header when making requests to protected resources.

**OAuth 2.0: This is very popular authentication method which is being used most nowadays. We have seen so many times while installing an application at the time of sign-up it asks to sign up with Google account or by Facebook etc. if we choose Facebook then we just have to login our Facebook account and approve Yes to access. And through this process directly we can login without sign-up. So basically, it provides a token along with some information and that information & token get stored in that 3rd party application’s database. That is OAuth 2.0 method.**

**Ex:** Basically, what we are doing, we are delegating our authority to google then after that Google will ask you that like “Do you want to authorized this Instagram application?” If we say ‘Yes’ then it will give our information to Instagram. And that information will store in Instagram database. Next time it won’t ask login details to login. And if we change our password then automatically, we will be logged out from Instagram. Again, it will follow same process to login. It will send a token to Instagram that will be stored in Instagram’s database. If we remove our Instagram details from Google then it will take back all that information from Instagram.

“**OAuth 2.0 is designed only for authorization, for granting access to data and features from one application to another. OpenID Connect (OIDC) is a thin layer that sits on top of OAuth 2.0 that adds login and profile information about the person who is logged in. The OpenID Connect flow looks the same as OAuth.**” So basically, OIDC is one step ahead to OAuth 2.0