

API Testing – Software testing

API testing, or application programming interface testing, is a type of software testing that focuses on the

testing of individual API methods and the interactions between different APIs.

This type of testing is typically performed

**at the integration level,
after unit testing is completed, and
before user interface testing begins.**

It is used to validate that the API behaves correctly and that it meets the requirements of the system.

API testing can be performed manually or using automated testing tools. Some common tasks that are performed during API testing include:

- Testing the functionality of the API to ensure it behaves as expected
- Verifying that the API returns the correct response for different input values
- Checking for error handling and validation of input
- Testing for security vulnerabilities
- Checking for performance and scalability of the API
- API testing is important because it ensures that the different components of a system can communicate with each other correctly and that the system can handle a large volume of requests.

It is also used to ensure that the API is compatible with different platforms and operating systems, and can be integrated with other systems and applications.

API Testing : As we know API stands for Application Programming Interface which acts as an intermediate of communication between two applications. Due to this intermediary role of API (Application Programming Interface) two applications talk to each other and performs the required actions efficiently.

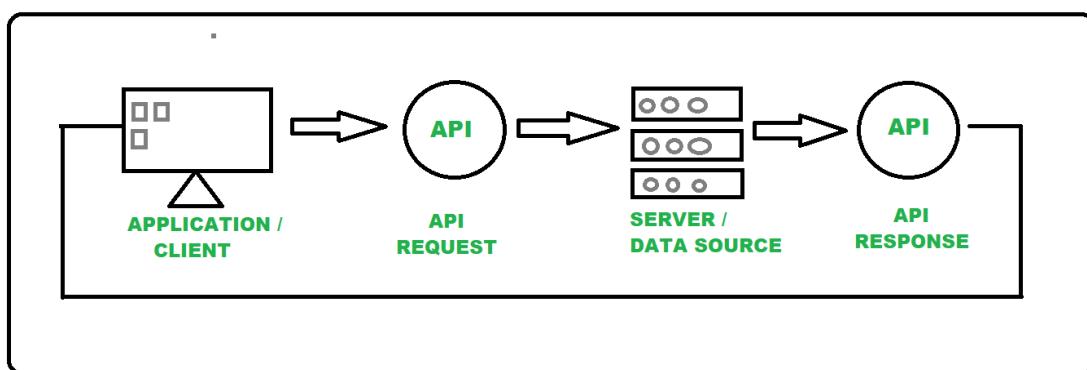
API contains a set of rules and guidelines based on which the applications are developed. So in simple we can say an API acts as an interface between two software applications so that two software applications can communicate with each other.

API testing typically includes the following steps:

- Reviewing the API documentation to understand the functionality and expected inputs and outputs
- Writing test cases that exercise the different functionality of the API
- Executing the test cases and comparing the expected results with the actual results
- Analysing the results and identifying any issues that need to be fixed

There are several types of API testing, including:

- **Functional testing:** Testing the functionality of the API to ensure it behaves as expected
- **Security testing:** Testing the security of the API to ensure it is protected against common vulnerabilities
- **Performance testing:** Testing the performance of the API to ensure it can handle the expected load
- **Interoperability testing:** Testing the compatibility of the API with other systems
- **Usability testing:** Testing the usability of the API for developers
- Tools such as Postman, SoapUI, and Runscope can be used to automate and simplify the process of API testing.



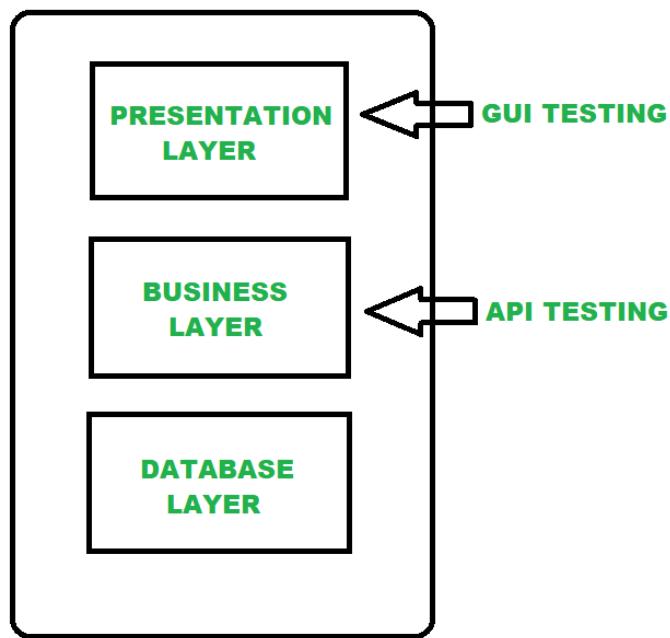
Types of API testing

API Testing refers to test the APIs which are used in the application just to validate that the APIs are working fine. When a system has a collection of APIs, these needs to be tested to know that the system is working perfectly or not. Mostly we can say that API testing confirms system's performance, reliability, security and functionality. **Below list represents some of the tools which are used for API Testing :**

- Postman
- Katalon Studio
- Soap UI
- Parasoft
- REST assured
- Tricentis Tosca

- Ping API
- Assertible

GUI testing is different from the API testing as GUI testing is present at Presentation layer where the API testing is present at Business layer. If we take an example of a typical app then API is the middle layer in between UI layer and Data base layer and due to this API communication and data exchange between the applications occur. The below **figure** represents the layer at which API testing is performed :



Layers of API Testing

API testing Types : There are multiple types of testing which are most often used as form of API testing which means during multiple types of testing simultaneously API can be tested. So below list represents the types of API testing i.e.

1. [Unit Testing](#)
2. [Integration Testing](#)
3. [End-to-End Testing](#)
4. [Performance Testing](#)
5. [Functional testing](#)
6. [Security Testing](#)
7. [Load testing](#)
8. [Penetration testing](#)
9. [Reliability testing](#)
10. [Fuzz testing](#)

What exactly we check during API testing :

- Data accuracy.

- Response time.
- Duplicate or missing functionality.
- Authorization checks.
- Multithreaded issues.
- Security and performance issues.
- Error codes if API returns.
- Reliability issues.

Benefits of API Testing :

Like we get a lot of advantages by using APIs in application, similarly by performing API testing we achieve a lot of things towards the success of the developed application. Below are some benefits i.e.

- Earlier validation of correctness in response and data.
- Earlier test maintenance.
- Better speed and coverage of testing.
- GUI independent testing.
- Reduced testing cost.
- Language independent test.
- Helpful in testing core functionality.
- API testing has several benefits that make it an important aspect of software testing:
- Improved functionality: API testing helps ensure that the functionality of the API is working as expected and that the data being exchanged is accurate and complete.
- Increased security: API testing helps identify and fix security vulnerabilities such as SQL injection and cross-site scripting. This helps ensure that the API is protected against common threats and that sensitive data is secure.
- Improved performance: API testing helps identify and fix performance bottlenecks, such as slow response times or high error rates. This helps ensure that the API can handle the expected load and that users have a positive experience when using it.
- Better integration: API testing helps ensure that the different systems that make up an application are working together correctly and that the data being exchanged is accurate and secure.
- Reduced risk: By identifying and fixing issues before the application is deployed to production, API testing helps reduce the risk of system failure or poor performance in production.
- Cost-effective: API testing is more cost-effective than fixing problems that occur in production. It is much cheaper to identify and fix issues during the testing phase than after deployment.
- Improved developer experience: By making sure that the API is easy to use, well-documented, and provides useful error messages, API testing helps improve the developer experience and encourage adoption.

- Greater flexibility: API testing allows teams to test the application without a user interface, which can be useful when testing microservices or when the user interface is not yet developed.

Disadvantages of API Testing:

API testing can have some disadvantages, including:

- Complexity: API testing can be complex, especially when testing multiple APIs or when testing APIs that are integrated with other systems.
- Limited Visibility: Since API testing is performed at the integration level, it can be difficult to see how the API is interacting with other components of the system. This can make it difficult to identify and troubleshoot issues.
- Security: APIs can introduce security vulnerabilities if they are not properly tested and secured. This can be a significant concern for organizations that handle sensitive data.
- Difficulty in testing non-functional requirements: Non-functional requirements such as performance, scalability and security are difficult to test with functional testing
- Time consuming: The time required to develop and execute test scripts for APIs can be longer than other types of testing.
- Limited documentation: Limited or poor documentation of the API can make it difficult for testers to understand how the API should behave.
- Limited test coverage: It is difficult to test all possible scenarios and edge cases with API testing.
- Cost: Automated API testing tools can be expensive and require a significant investment.

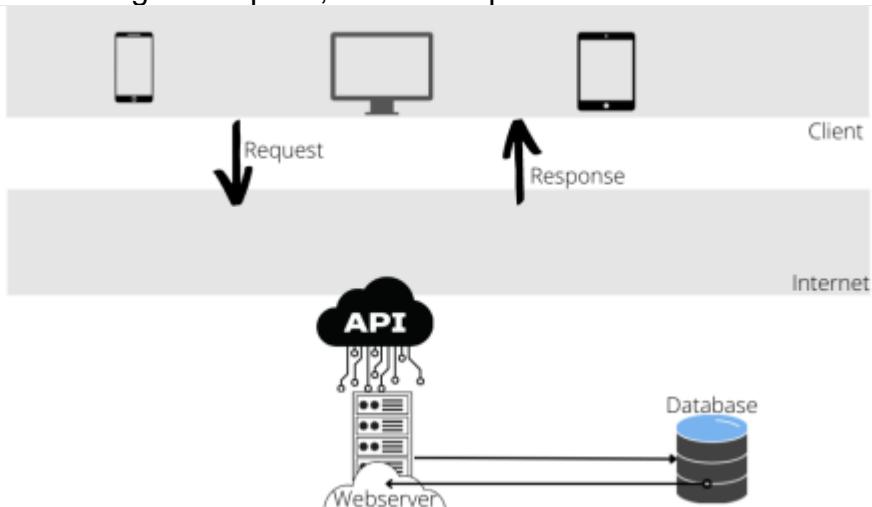
Types of Bugs that may occur in API Testing:

- Performance Issues – API response Time can be very high, and they may have latency.
- Response data may not structure correctly (JSON or XML)
- Security Issues
- Incorrect handling of valid argument values
- Improper errors/warning to caller
- Missing or Duplicate Functionality
- Reliability Issues : Difficulty in connecting and getting a response from API

What is API Testing?

API testing is integral to software development to provide optimal application performance. API, a specialized application programming interface, is a collection of protocols, tools, and functions that facilitate seamless communication between software applications.

Client/ client-side- includes all the devices used to interact with the application. The Internet is an enabler; it assists in transporting client requests to the Web server along with relevant details filled in by the user on the client side. Web-server/server-side is the holy place where all processing takes place. Once the processing is complete, the server pushes the data back to the client.

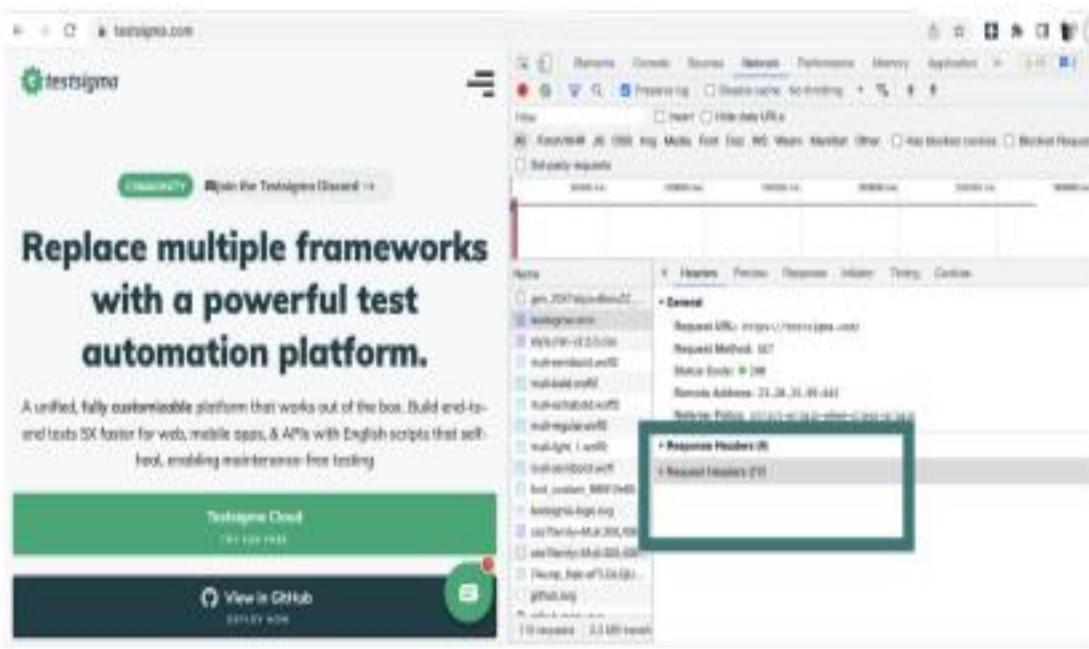


API Facets

APIs come with 2 broad facets

1. **Request**— An encapsulation of action performed along with pertinent details to be sent to the server.
2. **Response**— An encapsulation of processed data along with supplement details sent back to client by the server.

Both request and response come with header attached. Take a look at any network call, you should be able to see headers quite evidently.



The Request header passes all the additional details to the server to process the Request. In the example below, we see a set of keys and values. The key names are case insensitive.

Sometimes Request also carries payload details; in the below example, I am trying to join the [Testsigma discord community](#). To make that happen, I entered some details through UI and clicked on the “Send” button. In the backend call, we can see that the details passed to the server when the event is triggered fill the payload.

A web server processes the Request and prepares the response body and additional header details. For example, when I chose to join the discord community, the API responded with further information about filling captcha. We plan to perform regressive testing on the response body during [API testing](#) to verify that the sent details match our expectations.

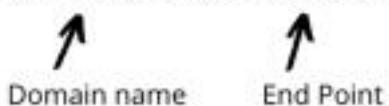
How are API called?

Each user interaction on UI is associated with an API that will be accessed through a URL. Let us say a user fills up a form and clicks on “Send” button, on trigger of this event respective API will be invoked. In the below picture we can see the Request URL is pointing to the endpoint ‘Signup’.

Screenshot of a browser developer tools Network tab showing a request to https://testsigma.com/signup. The Headers tab is selected. General details include:

- Request URL: https://testsigma.com/signup
- Request Method: GET
- Status Code: 200
- Remote Address: 23.20.25.99:443
- Referrer Policy: strict-origin-when-cross-origin

Request URL: https://testsigma.com/automated-web-application-testing



API Request Method

Every request is associated with the “Request Method”. This method specifies the desired action to be performed by the server. Below are the frequently used methods

- 1. GET:** This method is used to retrieve the details, this performs a read-only operation.
- 2. POST:** This method submits data by creating an entry into database. This method should be used wisely as it can change the state causing side effects if not handled properly.
- 3. PUT:** This method is used to update any data, the data that needs to be replaced is sent as part of payload.
- 4. DELETE:** This method just deletes the data.
- 5. HEAD:** This is similar to get method, however there is no response body sent.
- 6. PATCH:** This method specifies how to update the data, this can cause side effects if not handled with care.

Few FAQs about request methods

1. How will I get to know which API follows what method?

Ideally software development teams maintain API documentations. Every API is preceded with the respective method in the documentation.

2. Can one API endpoint perform multiple actions?

Yes, sometimes same endpoint is designed to do multiple methods.

3. Can I use API without specifying method?

No, you have to suffice the API call with request method

4. Can all APIs be accessed through URL

Yes and NO, some APIs are open for public and may not need authentication. However most of the business and enterprise APIs are behind API gateway and one may have to authenticate before calling the API.

5. Can same endpoint have multiple method associated?

Yes, it is completely possible to define multiple actions/verbs for same endpoint. Hence it is important to know beforehand what actions are supported by the API.

API Response

Every API request is associated with response. The response has 2 facets

1. Status code : Numeric representation of the web server response. There are predefined set of status code which can be reused. Or, teams can create their own status code as per convenience. However, appropriate status code range should be used.

INFORMATIONAL RESPONSES	(100-199)
SUCCESSFUL RESPONSES	(200-299)
REDIRECTION MESSAGES	(300-399)
CLIENT ERROR RESPONSES	(400-499)
SERVER ERROR RESPONSES	(500-599)

2. Status Message : Each status code is associated with detailed response message.

for example:

Status code 200 → Success

Status code 401 → Unauthorized

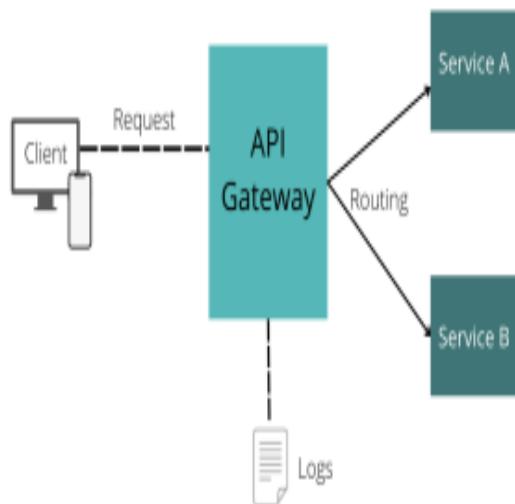
Status code 404 → Not found

API Gateway

API usage is not as simple in real world, when companies are hosting large scale APIs protecting them from misuse is their primary concern. API gateway helps establishing authentication to verify the calls before transferring it to further execution. The

capability of API gateway is not just restricted to authentication, it also provides multitude of services like

- I. Routing
- ii. Rate Limiting
- iii. Analytics
- iv. Security
- v. Policies etc



API Authentication

Since APIs deal with protected resources, the request processing should be backed with authentication process to ensure the access is provided only to the intended user. There is a difference between authentication and authorisation.

Authentication is a process of **verifying** the **identity** of the user it solely answers who you are, whereas **authorisation** mostly deals with **access management** and comes into play only after the user is identified and verified successfully.



Most commonly used authentication are

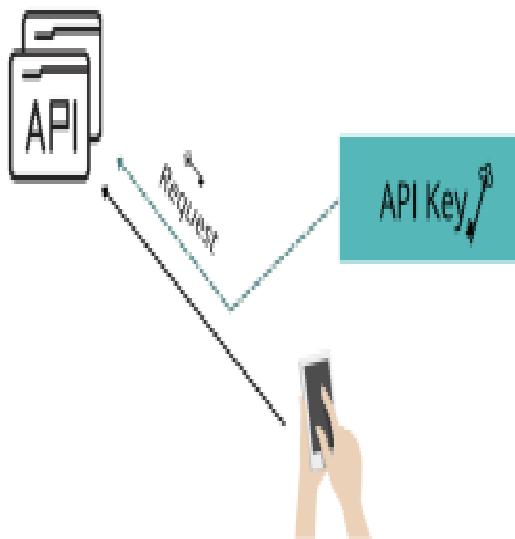
1. Basic Authentication:

This technique involve providing username and password for user verification. Authentication sounds little flimsy isn't it?, but this is how it is going to work. When user enters their credentials the details are encoded in Base64 generating a Key which will be bundled in request header and sent to server for verification. Server Verifies the key with the stored username and password. If the identity is verified the request is fulfilled else an error is sent back denying request sent.



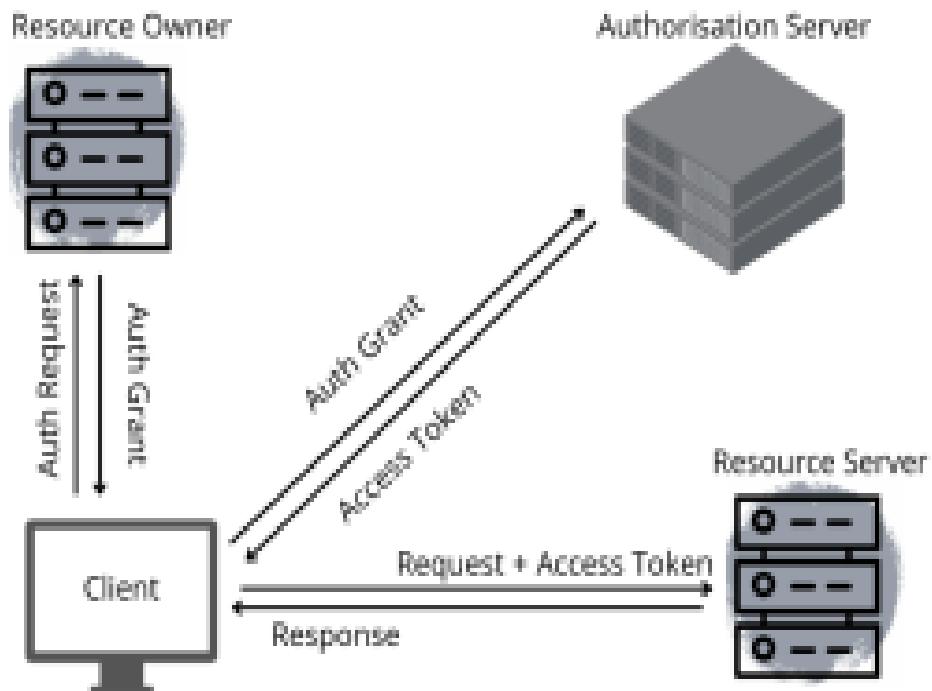
2. API Key Authentication :

API key is a long encrypted string which identifies the application without any principal. These are sent either as a part of request header or URL. When client recognises the API key server will process the request.



3. OAUTH Authentication :

This technique is considered quite powerful and secure way of authenticating the users. oAuth technique can also be used for authorisation. Initially a user may have to login to the oAuth application using the credentials to generate a token. The generated token is attached as part of request header, which will be sent to authentication server in order to verify. If the token is recognised the API request will be processed



API testing basics for beginners

Generating Test cases for API testing

API testing is a [black box testing](#) technique, one don't really delve into what is implemented within an API. Having said that, if you would like to particularly walk through the code to test the API that is fine too. The main task in [testing APIs](#) is to figure out the [test cases](#) just like any other tests out there.

For example consider the below [REST API](#)

GET <https://gorest.co.in/public/v2/users>

Looking at the API method we understand that this is going to perform a read operation and fetch the details requested which is list of all users. The response returned will be in the JSON format. REST Apis generally use JSON structure to represent the response body.

Let us go ahead churning the test cases for this API

1. On Successful execution the API should return status code 200 (message : Success)
2. Specific user details can be fetched by passing user id in URL (<https://gorest.co.in/public/v2/users/25>)

3. The JSON response should match the below schema

```
{  
  "id":integer,  
  "name":string,  
  "email":string,  
  "gender":string,  
  "status":string  
}
```

4. Wrong URL should respond with 404 (Not found)

5. Multiple User ID in the URL should respond with 404 error

API Testing Approach

There are various approaches to API testing, and the following points highlight some of the standard testing approaches:

1. Functional testing: This involves testing the functionality of the API to ensure that it meets the specified requirements.
2. Security testing: This involves testing the security of the API to ensure that it is not vulnerable to security threats.
3. Load testing: This involves testing the performance of the API under different loads to ensure that it can handle the expected traffic.
4. Integration testing: This involves testing the integration of the API with other software components to ensure that it works seamlessly.

Overall, a comprehensive API testing approach should cover all these testing approaches to ensure the quality and reliability of the software application.

How to Test API?

- Understand the API's Functionality

Before you start testing an API, you need to understand its functionality. This includes the data it exchanges with the client, the expected format of the data, the input parameters, and the expected output. Understanding the API's functionality will help you design your test strategy and ensure that your test cases cover all the required scenarios.

- Test the API Endpoints

Once you understand the API well, you can begin testing the endpoints. This involves sending requests to the API using various HTTP methods (GET, POST, PUT, DELETE).

Best Practices of API Testing

The best practices of API testing are essential to ensure the functionality and reliability of an application programming interface.

1. Plan and design the API tests before starting the testing process.
2. Use a testing framework that provides robust features to automate API testing.
3. Create test data covering a wide range of scenarios and edge cases.
4. Verify the API's functional behavior, security, and performance.
5. Conduct regression testing to ensure that changes and updates do not impact the existing API functionality.
6. Integrate API testing with continuous integration and delivery pipelines.
7. Collaborate with developers and other stakeholders to identify and resolve issues early.
8. Use monitoring tools to track API performance and identify issues before they impact end-users.

Challenges of API Testing

Here are some of the most common challenges:

1. Complex API architecture and integration with other systems can make testing difficult and time-consuming.
2. API testing requires programming skills and knowledge of HTTP protocols and methods.
3. Test data management can be challenging, especially when dealing with large data sets.
4. Ensuring API security and authentication is essential, but testing can be challenging.
5. API testing requires close collaboration between developers, testers, and stakeholders to ensure that the APIs meet the business requirements and user expectations.

Conclusion

Hey, it's time to wrap things up! API testing is super important when it comes to software development. It ensures that APIs play nice with other systems and do what they should. If you're new to the game, getting a handle on the API testing basics,

including the different tests, tools, and tricks involved, is key. This guide should give you a solid overview of API testing basics, so you can confidently start testing your APIs. But listen up: API testing isn't a one-and-done deal; it's an ongoing process that you must keep up with to keep your APIs in tip-top shape. Keep learning, keep testing, and keep improving those APIs!

[Try out Testsigma for our API test automation needs](#)

Happy Testing !

Frequently Asked Questions

How will I get to know which API follows what method?

The request header carries all the supplemental details to be passed on to the server to process the Request. In the example below, we see a set of keys and values. The key names are case insensitive.

Can one API endpoint perform multiple actions?

Yes, sometimes the same endpoint is designed to do multiple methods.

Can I use API without specifying the method?

No, you have to suffice the API call with the request method

Can all APIs be accessed through a URL?

Yes, and NO, some APIs are open to the public and may not need authentication.

However, most business and enterprise APIs are behind API gateway, and one may have to authenticate before calling the API.

Can the same endpoint have multiple methods associated?

It is possible to define multiple actions/verbs for the same endpoint. Hence, it is essential to know beforehand what actions the API supports.

What are the three layers of API testing?

People commonly refer to the three layers of API testing as unit testing, integration testing, and end-to-end testing.

What is the process of API testing?

API testing typically involves identifying the requirements for the API, designing test cases, executing the tests, and analyzing the results.

What is Postman API?

Postman API is a platform that promotes the development, testing, and documentation of APIs.

API testing typically involves the following practices:

- **Unit testing:** To test the functionality of individual operation
- **Functional testing:** To test the functionality of broader scenarios by using a block of unit test results tested together
- **Load testing:** To test the functionality and performance under load
- **Runtime/Error Detection:** To monitor an application to identify problems such as exceptions and resource leaks
- **Security testing:** To ensure that the implementation of the API is secure from external threats
- **UI testing:** It is performed as part of end-to-end integration tests to make sure every aspect of the user interface functions as expected
- **Interoperability and WS Compliance testing:** Interoperability and WS Compliance Testing is a type of testing that applies to SOAP APIs. Interoperability between SOAP APIs is checked by ensuring conformance to the [Web Services Interoperability profiles](#). [WS-*](#) compliance is tested to ensure standards such as WS-Addressing, WS-Discovery, WS-Federation, WS-Policy, WS-Security, and WS-Trust are properly implemented and utilized
- **Penetration testing:** To find vulnerabilities of an application from attackers
- **Fuzz testing:** To test the API by forcibly input into the system in order to attempt a forced crash

HTTP Protocol

HTTP stands for **Hyper Text Transfer Protocol**. Communication between client and web servers is done by **sending HTTP Requests** and **receiving HTTP Responses**.

HTTP Protocol is used to perform

CRUD operations (Create, Read, Update, Delete) by sending **HTTP requests** with different **HTTP methods**.

HTTP methods are also called as **HTTP verbs**.

HTTP Request Structure

Every HTTP request should have a **URL** or **URI address** and a **method**.

(Highlighted in purple)
Example URI

`http://api.example.com/products?
name=laptop&available=true`

The format of a request message includes

- A request-line
- Zero or more header field(s) followed by CRLF (Carriage Return, Line Feed)
- An empty line
- A message-body (optional)

HTTP Methods in REST

REST
REpresentational State Transfer

It is a set of architectural principles for designing web services.

Web services based on REST Architecture are known as **RESTful web services**.

REST makes it easy to share data between clients and servers.

REST applications use HTTP methods like **GET, POST, DELETE, PUT**, etc., to do CRUD operations.

Categories of HTTP Methods

You can divide HTTP methods into two main categories :

- Safe HTTP Methods
- Idempotent Methods

Safe HTTP Methods

It doesn't change data on the server. It always returns the same response, no matter how many times it gets called.

Example - **GET, HEAD**

Idempotent Methods

Example - **GET, HEAD, PUT, DELETE, TRACE**

It may change data on the server. It always returns the same response, no matter how many times it gets called.

All safe methods are also idempotent, but not all idempotent methods are safe.

The 9 HTTP Methods

- 
- GET Method
 - POST Method
 - PUT Method
 - PATCH Method
 - DELETE Method
 - HEAD Method
 - OPTIONS Method
 - TRACE Method
 - CONNECT Method

Let us now see them in detail.

GET Method

A GET Request is used to **request information** from a resource such as a website, a server, or an API.

Example ↗

`GET /api/employees/{employee-id}`

Returns a specific employee by employee id.

`GET /api/employees`

Returns a list of all employees.

Since the GET method should never change the data on the resources and just read them(read-only), it is considered a **safe method**.

The GET method is also **idempotent**.

Test an API with a GET Method

When we want to test an API, the **most popular method** that we would use is the GET method.

Therefore, We expect the following to happen ↗

- If the resource is accessible, the API returns the **200 Status Code**, which means **OK**.
- Along with the 200 Status Code, the server usually returns a **response body** in XML or JSON format. So, for example, we expect the [GET] /members endpoint to return a list of members in XML or JSON.
- If the server does not support the endpoint, the server returns the **404 Status Code**, which means **Not Found**.
- If we send the request in the wrong syntax, the server returns the **400 Status Code**, which means **Bad Request**.

GET Requests

- When you load a website. That's a GET request
- It's a request to get data from another computer
- You're simply asking for data and you're not asking to perform a task
- You're not creating, updating or deleting data
- Most common request type

Sending data through url

Caching: GET requests can be cached by browsers and proxy servers, which can improve performance and reduce server load.

Security: While GET requests are generally considered safe, sensitive data should not be passed in the URL query string to avoid exposing it in server logs or browser history.

Sure, here are some examples of GET requests:

1. Retrieve a List of Resources:

- Request: `GET /api/users`
- Response: Returns a list of users in JSON format.

2. Fetch a Single Resource by ID:

- Request: `GET /api/users/123`
- Response: Returns the user with ID 123 in JSON format.

3. Filter Data Based on Parameters:

- Request: `GET /api/products?category=electronics`
- Response: Returns a list of products belonging to the "electronics" category in JSON format.

4. Pagination:

- Request: `GET /api/posts?page=2&limit=10`
- Response: Returns the second page of posts with a limit of 10 posts per page in JSON format.

5. Sorting:

- Request: `GET /api/posts?sort=created_at`
- Response: Returns a list of posts sorted by the creation date in ascending order in JSON format.

```
from flask import Flask, jsonify, request
app = Flask(__name__)

# Dummy data for demonstration
users = [
    {"id": 1, "name": "John"},
    {"id": 2, "name": "Alice"},
    {"id": 3, "name": "Bob"}
]

products = [
    {"id": 1, "name": "Laptop", "category": "electronics"},
    {"id": 2, "name": "Smartphone", "category": "electronics"},
    {"id": 3, "name": "Headphones", "category": "electronics"},
    {"id": 4, "name": "Boo", "category": "books"},
    {"id": 5, "name": "TV", "category": "electronics"}
]

posts = [
    {"id": 1, "title": "Post 1", "created_at": "2024-04-15"}, {"id": 2, "title": "Post 2", "created_at": "2024-04-14"}, {"id": 3, "title": "Post 3", "created_at": "2024-04-13"}]

@app.route('/api/users/<int:user_id>', methods=['GET'])
def get_user(user_id):
    user = next((user for user in users if user['id'] == user_id), None)
    if user:
        return jsonify(user)
    else:
        return jsonify({'message': 'User not found'}), 404

@app.route('/api/products', methods=['GET'])
def get_products():
    category = request.args.get('category')
    if category:
        filtered_products = [product for product in products if product['category'] == category]
        return jsonify(filtered_products)
    else:
        return jsonify(products)

@app.route('/api/posts', methods=['GET'])
def get_posts():
    # Pagination
    page = int(request.args.get('page', 1))
    limit = int(request.args.get('limit', 10))
    start_index = (page - 1) * limit
    end_index = start_index + limit
    paginated_posts = posts[start_index:end_index]

    # Sorting
    sort_by = request.args.get('sort')
    if sort_by:
        paginated_posts = sorted(paginated_posts, key=lambda x: x[sort_by])
    return jsonify(paginated_posts)

if __name__ == '__main__':
    app.run(debug=True)
```

This code sets up routes for four different GET requests:

1. Retrieving a list of users (/api/users).
2. Retrieving a single user by ID (/api/users/<user_id>).
3. Filtering products by category (/api/products).
4. Retrieving paginated and sorted posts (/api/posts).

POST Method

It creates a new resource on the backend (server). We send data to the server in the request body.

Example:

POST /api/employees/department

Creates a department resource.

POST /api/employees/232/department/114/department-items

Creates a department item using the employee id and department id.

Two identical POST requests will create two new equivalent resources with the same data and different resource ids. We don't get the same result every time.

It is neither a safe nor an idempotent method.

Testing a POST Endpoint

Since the POST method creates data, we must be cautious about changing the data. Testing all the POST methods in APIs is highly recommended.

Here are some suggestions that we can do for testing APIs with POST methods:

- Create a resource with the POST method, and it should return the 201 Status Code.
- Perform the GET method to check if it created the resource was successfully created. You should get the 200 status code, and the response should contain the created resource.
- Perform the POST method with incorrect or wrong formatted data to check if the operation fails.

POST Requests

- Do not go through the standard URL, but use a URL as the endpoint
- Ask another computer to create a new resource
- Returns data about the newly created resource

Example 1: Create a New User	Example 3: Submit a Form
<p>Request:</p> <ul style="list-style-type: none">• Method: POST• URL: http://localhost:5000/api/users• Body: JSON format containing user data <p>Flask Code:</p> <pre>from flask import Flask, request, jsonify app = Flask(__name__) @app.route('/api/users', methods=['POST']) def create_user(): data = request.json users.append(data) message = "User created successfully" return jsonify({'message': message}), 201 if __name__ == '__main__': app.run(debug=True)</pre>	<p>Request:</p> <ul style="list-style-type: none">• Method: POST• URL: http://localhost:5000/api/submit_form• Body: Form data containing user input <p>Flask Code:</p> <pre>from flask import Flask, request, jsonify app = Flask(__name__) @app.route('/api/submit_form', methods=['POST']) def submit_form(): name = request.form.get('name') email = request.form.get('email') message = "Form submitted successfully" return jsonify({'message': message}), 201 if __name__ == '__main__': app.run(debug=True)</pre>

Submitting any form or field ,creating user in fb by filling form

1. **Definition**: The POST method is used to submit data to be processed to a specified resource. It's commonly used for creating new resources.

3. **Request Body**: Data to be sent to the server is included in the body of the request. This can be in various formats like JSON, XML, or form data.

4. **Security**: POST requests can contain sensitive data, so it's important to use HTTPS to encrypt the data in transit. Additionally, input validation and sanitization should be performed on the server-side to prevent security vulnerabilities like SQL injection and XSS attacks.

8. **Examples**: Provide examples of POST requests for creating resources, such as creating a new user, adding a product to a database, or submitting a form.

PUT Method

Using this, we can **update** an **existing resource** by sending the updated data as the content of the request body to the server.

Example ↗

PUT /api/employees/123

Update employee by employee id

If it applies to a collection of resources, it **replaces the whole collection**, so be careful using it. The server will return the **200 or 204 status codes** after updating.

The PUT method is **idempotent** but **not safe**.

Test an API with a PUT Method

The PUT method is **idempotent**, and it modifies the entire resources.

Make sure to do the following operations ↗

- Send a **PUT request** to the server many times, and it should always return the **same result**.
- When the server completes the PUT request and updates the resource, the response should come with **200 or 204 status codes**.
- After the server completes the PUT request, make a **GET request** to check if the data is **updated correctly** on the resource.
- If the input is **invalid** or has the **wrong format**, the resource must not be updated.

1. **Definition**: The PUT method is used to update a resource or create it if it doesn't exist at a specified URL.

3. **Request Body**: Data to be updated or created is included in the body of the request. This can be in various formats like JSON, XML, or form data.

4. **Idempotent**: PUT requests are idempotent, meaning making the same request multiple times should have the same result. If the resource already exists, it will be updated with the provided data. If it doesn't exist, a new resource will be created.

5. **Security**: Similar to POST requests, PUT requests can contain sensitive data, so it's important to use HTTPS to encrypt the data in transit. Input validation and sanitization should be performed on the server-side to prevent security vulnerabilities.

7. **Examples**: Provide examples of PUT requests for updating resources, such as updating a user's profile information, modifying product details, or editing existing data.

Example 1: Update User Profile	Example 2: Modify Product Details	Example 3: Edit Existing Data
<p>Request:</p> <ul style="list-style-type: none">Method: PUTURL: http://localhost:5000/api/users/123 (replace 123 with the user ID)Body: JSON format containing updated user data	<p>Request:</p> <ul style="list-style-type: none">Method: PUTURL: http://localhost:5000/api/products/1 (replace 1 with the product ID)Body: JSON format containing updated product data	<p>Request:</p> <ul style="list-style-type: none">Method: PUTURL: http://localhost:5000/api/edit_dataBody: JSON format containing data to be edited
<p>Flask Code:</p> <pre>from flask import Flask, request, jsonify app = Flask(__name__) users = [{"id": 123, "name": "John", "email": "john@example.com"}] # Dummy data for demonstration @app.route('/api/users/<int:user_id>', methods=['PUT']) def update_user(user_id): data = request.json for user in users: if user['id'] == user_id: user.update(data) return jsonify({'message': 'User updated successfully'}), 200 return jsonify({'message': 'User not found'}), 404</int:user_id></pre>	<p>Flask Code:</p> <pre>from flask import Flask, request, jsonify app = Flask(__name__) products = [{"id": 1, "name": "Laptop", "price": 1000}] # Dummy data for demonstration @app.route('/api/products/<int:product_id>', methods=['PUT']) def update_product(product_id): data = request.json for product in products: if product['id'] == product_id: product.update(data) return jsonify({'message': 'Product updated successfully'}), 200 return jsonify({'message': 'Product not found'}), 404</int:product_id></pre>	<p>Flask Code:</p> <pre>from flask import Flask, request, jsonify app = Flask(__name__) data = {"key1": "value1", "key2": "value2"} # Dummy data for demonstration @app.route('/api/edit_data', methods=['PUT']) def edit_data(): data_to_edit = request.json for key, value in data_to_edit.items(): if key in data: data[key] = value return jsonify({'message': 'Data edited successfully'}), 200 if __name__ == '__main__': app.run(debug=True)</pre>

PATCH Method

Similar to PUT, PATCH updates a resource, but it **updates data partially** and not entirely.

Example

```
PATCH /api/employees/123
{
  "name" : "Brij"
}
Updates name for employee id 123.
```

The PATCH method **updates the provided fields** of the employee entity. In general, this modification should be in a standard format like JSON or XML.

It is **neither a safe nor an idempotent method**

Test an API with a PATCH Method

To test an API with the PATCH method, follow the steps for the testing API with the **PUT** and the **POST** methods.

Consider the following results

- Send a PATCH request to the server. The server will return the **2xx HTTP status code**, which means, the request is successfully received, understood, and accepted.
- Perform the **GET request** and verify that the content is updated correctly.
- If the **request payload** is incorrect or ill-formatted, the operation must fail.

The PATCH method in API is similar to the PUT method, but it's typically used for partial updates to an existing resource, rather than replacing the entire resource. Here's an explanation of the PATCH method:

- Partial Updates**: PATCH requests are used when you want to update only specific fields or properties of a resource, rather than replacing the entire resource. This can be useful when you only need to modify certain attributes of an object without affecting others.
- HTTP Verb**: Like PUT and POST, PATCH is an HTTP method used for making requests to a server. It's typically used in situations where a PUT request might be too heavy-handed, especially when dealing with large or complex resources.
- Request Body**: Data to be updated is included in the body of the PATCH request. This data typically contains only the fields that need to be modified, rather than the entire resource representation.
- Idempotent**: PATCH requests are not necessarily idempotent. While they can be designed to be idempotent in certain cases, it's not a requirement like it is for PUT requests.
- Response**: After processing the PATCH request, the server responds with a status code indicating the success or failure of the operation, similar to other HTTP methods.
- Examples**: PATCH requests can be used for various scenarios, such as updating a user's profile picture, changing the status of a task, or modifying specific attributes of a product.
- Best Practices**: It's important to design PATCH requests carefully to ensure that they accurately reflect the intended partial updates and that they're implemented in a way that is consistent with the application's data model and business logic.

In summary, the PATCH method is used for making partial updates to existing resources in a flexible and efficient manner, making it a valuable tool in API design and development.

Example 1: Update User's Profile Picture	Example 3: Modify Product Attributes	Example 2: Change Task Status
Request:	Request:	Request:
<ul style="list-style-type: none"> Method: PATCH URL: http://localhost:5000/api/users/123 (replace 123 with the user ID) Body: JSON format containing the new profile picture URL 	<ul style="list-style-type: none"> Method: PATCH URL: http://localhost:5000/api/products/789 (replace 789 with the product ID) Body: JSON format containing the new product attributes to be updated 	<ul style="list-style-type: none"> Method: PATCH URL: http://localhost:5000/api/tasks/456 (replace 456 with the task ID) Body: JSON format containing the new task status
Flask Code:	Flask Code:	Flask Code:
<pre>from flask import Flask, request, jsonify app = Flask(__name__) users = [{"id": 123, "name": "John", "profile_picture": "old_url.jpg"}] # Dummy data for demonstration</pre>	<pre>from flask import Flask, request, jsonify app = Flask(__name__) products = [{"id": 789, "name": "Laptop", "price": 1000}] # Dummy data for demonstration</pre>	<pre>from flask import Flask, request, jsonify app = Flask(__name__) tasks = [{"id": 456, "title": "Task 1", "status": "pending"}] # Dummy data for demonstration @app.route('/api/tasks/<int:task_id>', methods=['PATCH']) def change_task_status(task_id): data = request.json for task in tasks: if task['id'] == task_id: task['status'] = data.get('status') return jsonify({'message': 'Task status updated successfully'}), 200 return jsonify({'message': 'Task not found'}), 404 if __name__ == '__main__': app.run(debug=True)</int:task_id></pre>

The main difference between the PUT and PATCH methods in API lies in how they are used for updating resources:

1. **PUT Method**:

- The PUT method is used to update or replace an entire resource at a specific URL.
- When making a PUT request, the client sends a complete representation of the resource, including all fields, even those that are not being updated.
- If a resource exists at the specified URL, the entire resource is replaced with the new representation provided in the request body.
- PUT requests are idempotent, meaning making the same request multiple times should have the same result.

2. **PATCH Method**:

- The PATCH method is used to make partial updates to an existing resource.
- When making a PATCH request, the client sends only the fields that need to be updated, rather than the entire resource representation.
- If a resource exists at the specified URL, only the specified fields in the request body are updated, leaving the rest of the resource unchanged.
- PATCH requests are not necessarily idempotent, as subsequent requests may produce different results based on the initial state of the resource and the applied patches.

In summary, PUT is typically used for full updates where the client sends the complete representation of the resource, while PATCH is used for partial updates where the client sends only the modified fields.

DELETE Method

The DELETE method **deletes a resource**. Regardless of the number of calls, it returns the **same result**.

Example

`DELETE /api/employees/235`

Delete employee by employee id.

Most APIs always return the **200 status code** even if we try to delete a deleted resource but in some APIs, If the target **data no longer exists**, the method call would return a **404 status code**.

The DELETE method is **idempotent** but **not safe**.

Testing a DELETE Endpoint

When it comes to deleting something on the server, we should be **cautious**. We are deleting data, and it is **critical**.

Then perform the following actions

- Call the **POST method** to create a new resource. Never test **DELETE** with **actual data**. For example, first, create a new employee and then try to delete the employee you just created.
- Make the **DELETE request for a specific resource**. For example, the request `[DELETE] /employees/{employee-id}` deletes a employee with the specified employee id.
- Call the **GET method** for the deleted employee, which should return **404**, as the resource no longer exists.

The DELETE method in API is used to request the removal of a resource from the server at a specified URL. Here's an overview of the DELETE method:

1. **HTTP Verb**: DELETE is one of the standard HTTP methods, alongside GET, POST, PUT, and PATCH.
2. **Resource Removal**: DELETE requests are used to instruct the server to remove the resource identified by the URL from its storage.
3. **Idempotent**: DELETE requests are idempotent, meaning making the same request multiple times should have the same result. Once a resource is deleted, subsequent DELETE requests on the same resource will have no effect.
4. **Response**: After processing the DELETE request, the server typically responds with a status code indicating the success or failure of the operation. Common status codes include 200 (OK) for successful deletion and 404 (Not Found) if the resource does not exist.
5. **Security**: DELETE requests can have significant consequences as they permanently remove data from the server. Proper authentication and authorization mechanisms should be in place to ensure that only authorized users can perform DELETE operations.
6. **Examples**: DELETE requests are commonly used in scenarios such as deleting a user account, removing a file from a server, or deleting a record from a database.

Example 1: Delete User Account	Example 2: Remove File from Server	Example 3: Delete Record from Database
<p>Request:</p> <ul style="list-style-type: none"> Method: DELETE URL: http://localhost:5000/api/users/123 (replace 123 with the user ID) 	<p>Request:</p> <ul style="list-style-type: none"> Method: DELETE URL: http://localhost:5000/api/files/image.jpg (replace image.jpg with the file name) 	<p>Request:</p> <ul style="list-style-type: none"> Method: DELETE URL: http://localhost:5000/api/products/789 (replace 789 with the product ID)
<p>Flask Code:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) users = [{"id": 123, "name": "John"}, {"id": 456, "name": "Alice"}, {"id": 789, "name": "Bob"}] @app.route('/api/users/<int:user_id>', methods=['DELETE']) def delete_user(user_id): global users for i, user in enumerate(users): if user['id'] == user_id: del users[i] return jsonify({'message': 'User deleted successfully'}), 200 if __name__ == '__main__': app.run(debug=True)</int:user_id></pre>	<p>Flask Code:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) @app.route('/api/files/<filename>', methods=['DELETE']) def delete_file(filename): file_path = os.path.join(app.root_path, 'uploads', filename) if os.path.exists(file_path): os.remove(file_path) return jsonify({'message': 'File deleted successfully'}), 200 else: return jsonify({'message': 'File not found'}), 404 if __name__ == '__main__': app.run(debug=True)</filename></pre>	<p>Flask Code:</p> <pre>from flask import Flask, jsonify app = Flask(__name__) products = [{"id": 789, "name": "Laptop", "price": 1000}, {"id": 087, "name": "Phone", "price": 800}] @app.route('/api/products/<int:product_id>', methods=['DELETE']) def delete_product(product_id): global products for product in products: if product['id'] == product_id: products.remove(product) return jsonify({'message': 'Product deleted successfully'}), 200 return jsonify({'message': 'Product not found'}), 404 if __name__ == '__main__': app.run(debug=True)</int:product_id></pre>

HEAD Method

The HEAD method is similar to the GET method. But it **doesn't have any response body**, so if it mistakenly returns the response body, it must be ignored.

Example

HEAD /api/employees

Similar to GET, but it does not return the list of employees.

Before requesting the GET endpoint, we can make a **HEAD request** to determine the size (Content-length) of the file or data that we are downloading.

The HEAD method is **safe** and **idempotent**.

Testing a HEAD Endpoint

One of the advantages of the HEAD method is that we can test the server if it is **available** and **accessible** as long as the API supports it.

The API can be tested as follows

- It is **much faster** than the GET method because it has no response body.
- The **status code** we expect to get from the API is **200**.
- Before every other HTTP method, we can **first test API** with the HEAD method.

The HEAD method in API is similar to the GET method, but it's used to retrieve metadata about a resource rather than the resource itself. Here's an overview of the HEAD method:

- HTTP Verb**: HEAD is one of the standard HTTP methods, alongside GET, POST, PUT, DELETE, and PATCH.
- Metadata Retrieval**: HEAD requests are used to retrieve metadata about a resource, such as its headers, content length, content type, etc.
- Response**: When a server receives a HEAD request, it responds with the headers that would be returned for a corresponding GET request to the same URL, but without the actual content of the resource. This allows clients to retrieve important metadata without downloading the entire resource, which can be useful for checking resource availability, content characteristics, or server status.
- Idempotent**: HEAD requests are idempotent, meaning making the same request multiple times should have the same result. However, they do not affect the server's state or cause any changes to the requested resource.

5. ****Examples**:** HEAD requests are commonly used in scenarios where clients need to check the status or characteristics of a resource without actually downloading its content, such as checking if a file exists, verifying the last modified timestamp of a document, or determining the size of a file.

In summary, the HEAD method provides a lightweight way to retrieve metadata about resources, making it a useful tool for efficient communication between clients and servers.

Example 1: Check Resource Availability	Example 2: Get Resource Metadata
Request:	Request:
<ul style="list-style-type: none"> Method: HEAD URL: <code>http://localhost:5000/api/users/123</code> (replace 123 with the user ID) 	<ul style="list-style-type: none"> Method: HEAD URL: <code>http://localhost:5000/api/files/image.jpg</code> (replace <code>image.jpg</code> with the file name)
Flask Code:	Flask Code:
<pre>from flask import Flask, make_response app = Flask(__name__) users = [123: {"name": "John", "age": 30, "city": "New York"}, 456: {"name": "Alice", "age": 25, "city": "Los Angeles"} # Dummy data for demonstration @app.route('/api/users/<int:user_id>', methods=['HEAD']) def check_user_availability(user_id): if user_id in users: response = make_response() response.status_code = 200 response.headers['X-User-Exists'] = 'true' return response else: return make_response('User not found', 404)</pre>	<pre>from flask import Flask, make_response import os app = Flask(__name__) @app.route('/api/files/<filename>', methods=['HEAD']) def get_file_metadata(filename): file_path = os.path.join(app.root_path, 'uploads', filename) if os.path.exists(file_path): file_size = os.path.getsize(file_path) last_modified = os.path.getmtime(file_path) response = make_response() response.status_code = 200 response.headers['Content-Length'] = file_size response.headers['Last-Modified'] = last_modified return response else: return make_response('File not found', 404)</pre>
	
	<pre>from flask import Flask, jsonify app = Flask(__name__) data = { "name": "John", "age": 30, "city": "New York" } # Dummy data for demonstration @app.route('/api/data', methods=['GET', 'HEAD']) def get_data(): if app.current_request.method == 'HEAD': # Respond with headers only, no content response = make_response() response.headers['Content-Type'] = 'application/json' response.headers['Content-Length'] = len(jsonify(data).data) return response else: # Respond with full data return jsonify(data) if __name__ == '__main__': app.run(debug=True)</pre>

OPTIONS Method

This method is used to get information about the **possible communication options** (permitted HTTP methods) for the given URL or an **asterisk** to refer to the entire server.

Example

```
OPTIONS /api/main.html/1
Returns permitted HTTP method in this URL.

OPTIONS * HTTP/1.1
Returns all permitted methods
```

Various browsers widely use the OPTIONS method to check whether the CORS (Cross-Origin resource sharing) operation is restricted on the targeted API or not.

Testing an OPTIONS Endpoint

To try it, consider the following

Depending on whether the server supports the OPTIONS method, we can test the server for the times of **FATAL failure** with the OPTIONS method.

- Make an OPTIONS request and check the header and the status code that returns.
- Test the case of failure with a resource that doesn't support the OPTIONS method.

```
from flask import Flask, jsonify
app = Flask(__name__)

@app.route('/api/data', methods=['GET', 'POST', 'PUT', 'DELETE'])
def data():
    if app.current_request.method == 'OPTIONS':
        # Respond with information about supported methods
        response = jsonify({
            'methods': ['GET', 'POST', 'PUT', 'DELETE'],
            'description': 'This endpoint supports various HTTP methods.'
        })
        response.headers['Allow'] = ', '.join(['GET', 'POST', 'PUT', 'DELETE'])
        response.headers['Access-Control-Allow-Methods'] = ', '.join(['GET', 'POST', 'PUT', 'DELETE'])
        return response
    elif app.current_request.method == 'GET':
        # Handle GET request
        return jsonify({'message': 'GET request received'})
    elif app.current_request.method == 'POST':
        # Handle POST request
        return jsonify({'message': 'POST request received'})
    elif app.current_request.method == 'PUT':
        # Handle PUT request
        return jsonify({'message': 'PUT request received'})
    elif app.current_request.method == 'DELETE':
        # Handle DELETE request
        return jsonify({'message': 'DELETE request received'})
```

The OPTIONS method in API is used to retrieve information about the communication options available for a particular resource or server. Here's an overview of the OPTIONS method:

1. **HTTP Verb**: OPTIONS is one of the standard HTTP methods, alongside GET, POST, PUT, DELETE, and PATCH.
2. **Resource Information**: OPTIONS requests are used to inquire about the communication options available for a resource or server, such as supported HTTP methods, allowed headers, supported authentication methods, etc.
3. **Response**: When a server receives an OPTIONS request, it responds with a list of the supported HTTP methods, headers, and other relevant information for the requested resource. This allows clients to determine the available communication options and make appropriate requests.
4. **Usage**: OPTIONS requests are commonly used during preflight CORS (Cross-Origin Resource Sharing) requests, where the client sends an OPTIONS request to the server to determine if a cross-origin request is allowed.
5. **Idempotent**: OPTIONS requests are idempotent, meaning making the same request multiple times should have the same result. They do not cause any changes to the server's state or affect the requested resource.
6. **Examples**: OPTIONS requests can be used in various scenarios, such as determining the allowed methods for a particular endpoint, checking CORS policies, or retrieving server capabilities.

In summary, the OPTIONS method provides a standardized way for clients to inquire about the communication options available for a resource or server, making it a useful tool for understanding server capabilities and making appropriate requests.

It's not mandatory to have an OPTIONS method implementation for every endpoint in your API. The OPTIONS method is primarily used for providing information about the communication options available for a particular resource or server.

However, if your API is designed to support CORS (Cross-Origin Resource Sharing), the OPTIONS method becomes more relevant. In CORS, browsers may make a preflight OPTIONS request to check if the server allows the intended cross-origin request. In such cases, you should handle OPTIONS requests appropriately to provide the required CORS headers and information about supported methods.

In general, if your API doesn't require CORS support or if there are no specific communication options that need to be exposed to clients, you may not need to implement the OPTIONS method for all endpoints. It's typically implemented for endpoints where it's necessary to provide information about supported methods or CORS policies.

TRACE Method

The TRACE method is for **diagnosis purposes**. It creates a **loop-back test** with the same request body that the client sent to the server before, and the successful response code is **200 OK**.

Example

TRACE /api/main.html

Responds the exact request that client sent.

The TRACE method could be **dangerous** because it could reveal credentials. A hacker could steal credentials, including internal authentication headers, using a **client-side attack**.

The TRACE method is **safe** and **idempotent**.

Test an API with a TRACE Method

- Make a standard HTTP request like a GET request to [/api/status](#)
- Replace GET with the TRACE and send it again.
- Check what the server returns. If the response has the same information as the original request, the **TRACE ability is enabled** in the server and **works correctly**.

The TRACE method in HTTP is used primarily for diagnostic purposes. When a server receives a TRACE request, it echoes back the received request to the client, allowing the client to see what changes, if any, were made by intermediate servers along the request chain.

However, the TRACE method is often disabled or restricted in modern web servers due to security concerns, as it can potentially expose sensitive information or be exploited in cross-site tracing (XST) attacks.

```
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/api/trace',
methods=['TRACE'])
def trace_request():
    # Echo back the received request
    request_info = {
        'method': request.method,
        'url': request.url,
        'headers':
            dict(request.headers),
        'data':
            request.get_data(as_text=True)
    }
    return jsonify(request_info)

if __name__ == '__main__':
    app.run(debug=True)
```

```
{
  "method": "TRACE",
  "url": "http://localhost:5000/api/trace",
  "headers": {
    "Host": "localhost:5000",
    "User-Agent": "python-requests/2.26.0",
    "Accept-Encoding": "gzip, deflate",
    "Accept": "*/*",
    "Connection": "keep-alive"
  },
  "data": ""
}
```

CONNECT Method

The CONNECT method is for making end-to-end connections between a client and a server. It makes a two-way connection like a tunnel between them.

Example

CONNECT www.example.com:443 HTTP/1.1

Connects to the URL provided.

For example, we can use this method to safely transfer a large file between the client and the server.

It is neither a safe nor an idempotent method

These are the 9 HTTP methods, their uses and a guide on how to test them. Hope you learned about them in detail.

DELETE Requests

- Do not go through the standard URL, but use a URL as the endpoint
- Ask another computer to delete a single resource or a list of resources
- **Use with caution**

Deleting anything

PATCH Requests

- Do not go through the standard URL, but use a URL as the endpoint
- Ask another computer to **update a piece** of a resource
- Are not fully supported by all browsers or frameworks, so we typically fall back on PUT requests
- Example: Updating a user's first name

Python Django does not support

PUT Requests

- Do not go through the standard URL, but use a URL as the endpoint
- Ask another computer to **update an entire** resource
- If the resource doesn't exist, the API might decide to **CREATE (CRUD)** the resource

HTTP Methods for RESTful Requests

HTTP Method	CRUD Operation	Example URL(s)
GET	Read	HTTP GET http://website.com/api/users/ HTTP GET http://website.com/api/users/1/
POST	Create	HTTP POST http://website.com/api/users/
DELETE	Delete	HTTP DELETE http://website.com/api/user/1/
PUT	Update/Replace	HTTP PUT http://website.com/api/user/1/
PATCH	Partial Update/Modify	HTTP PATCH http://website.com/api/user/1/

More details at <https://restfulapi.net/http-methods/>

JSON Example

```
{
  "key_val_example": "value",
  "array_example": [
    'array item 1',
    'array item 2',
  ],
  "object_example": {
    "key1": "value1",
    "key2": "value2"
  }
}
```



XML Example

```
<example>
  <field>
    Value
  </field>
  <secondField>
    Value
  </secondField>
  <nestedExample>
    <nestedField>
      Value
    </nestedField>
    <nestedSecondField>
      Value
    </nestedSecondField>
  </nestedExample>
</example>
```

Browsers use JavaScript for
their API requests.

Servers use any language that
runs on that computer.

Healthy Responses (2--)

- 200 — OK.
Request accepted.
- 201 — Created.
POST requests often return 201s when a resource is created.
- 202 — Accepted.
When a request is accepted but its not done processing.
Maybe the task goes into a queue.

Redirect Responses (3--)

- 301 — Moved Permanently.
When the endpoint has permanently changed. Update your endpoint.
- 302 — Found.
The endpoint you're accessing is temporarily moved to somewhere else.

Client Responses (4--)

- 400 — Bad Request.
Server cannot or will not process your request. Often this is due to malformed API keys or an invalid payload.
- 401 — Unauthorized.
You're not allowed here. Usually this is because you're missing authentication credentials (API keys)
- 403 — Forbidden.
The servers understands your request but won't execute it. Your API keys might not have the right permissions or you're trying to use an endpoint that you don't have access to.
- 404 — Not Found.
There's nothing here. Move along, move along.
- 405 — Method Not Allowed.
You're using the wrong HTTP Method. The endpoint might only accept GET requests and you might be POSTing to it, for example.

Server Responses (5--)

- 500 — Internal Server Error.
The server had a problem and couldn't process the request.
This is the only time you are out of control.

RESTful API Cheat Sheet

REpresentational State Transfer

Request Methods

- GET** { Used to get data only and does not modify the data at all
Should return 200, 400 or 404 responses
- POST** { Create a new resource (ie. creating a new user)
Should return 200, 201 or 204 responses
- DELETE** { Delete a resource (ie. delete a user)
Should return 200, 202 or 204 responses
- PUT** { Update a resource. If the resource doesn't exist,
the api might decide to create it and return a 201 response
Should return 200, 201 or 204 responses

@KalobTaulien

Common Status Codes

2-- Success Codes	
200	OK
201	Created
202	Accepted
204	No Content
4-- Client Error Codes	
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
405	Method Not Allowed
3-- Redirection Codes	
301	Moved Permanently
302	Found
5-- Server Error Codes	
500	Internal Server Error

Authentication

API keys are used as authentication credentials

CRUD Operations

Create, Read, Update, Delete

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Automation Testing' selected, showing collections like 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'History'. In the main area, under 'Library', there are three items: 'GET AddBook' (green), 'GET GetBook' (green), and 'GET DeleteBook' (green). The central part of the screen shows a 'POST AddBook' request. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2     "name": "Learn Appium Automation with Java",
3     "isbn": "ytntft",
4     "aisle": "227",
5     "author": "John doe"
6 }
```

Below the body, the 'Headers' tab is expanded, showing 'Content-Type: application/json'. At the bottom, the response status is shown as 'Status: 200 OK'.

Home Workspaces Reports Explore

Automation Testing New Import Overview Library POST AddBook GET GetBook POST DeleteBook No Environment

Collections + Library GET AddBook GET GetBook GET DeleteBook

APIs Environments Mock Servers Monitors History

Library / GetBook

GET https://rahulshettyacademy.com/Library/GetBook.php?ID=ytnft227

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> ID	ytnft227			
Key	Value	Description		

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```
1 [  
2   [  
3     "book_name": "Learn Appium Automation with Java",  
4     "isbn": "ytnft",  
5     "aisle": "227",  
6     "author": "John doe"  
7   ]  
8 ]
```

Status: 200 OK Time: 770 ms Size: 516 B Save Response

Find and Replace Console Bootcamp Runner Train Udemy

Home Workspaces Reports Explore

Automation Testing New Import Overview Library POST AddBook GET GetBook POST DeleteBook No Environment

Collections + Library GET AddBook GET GetBook GET DeleteBook

APIs Environments Mock Servers Monitors History

Library / DeleteBook

POST https://rahulshettyacademy.com/Library/DeleteBook.php

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Body none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  
2  
3 "ID" : "ytnft227"  
4  
5  
6
```

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

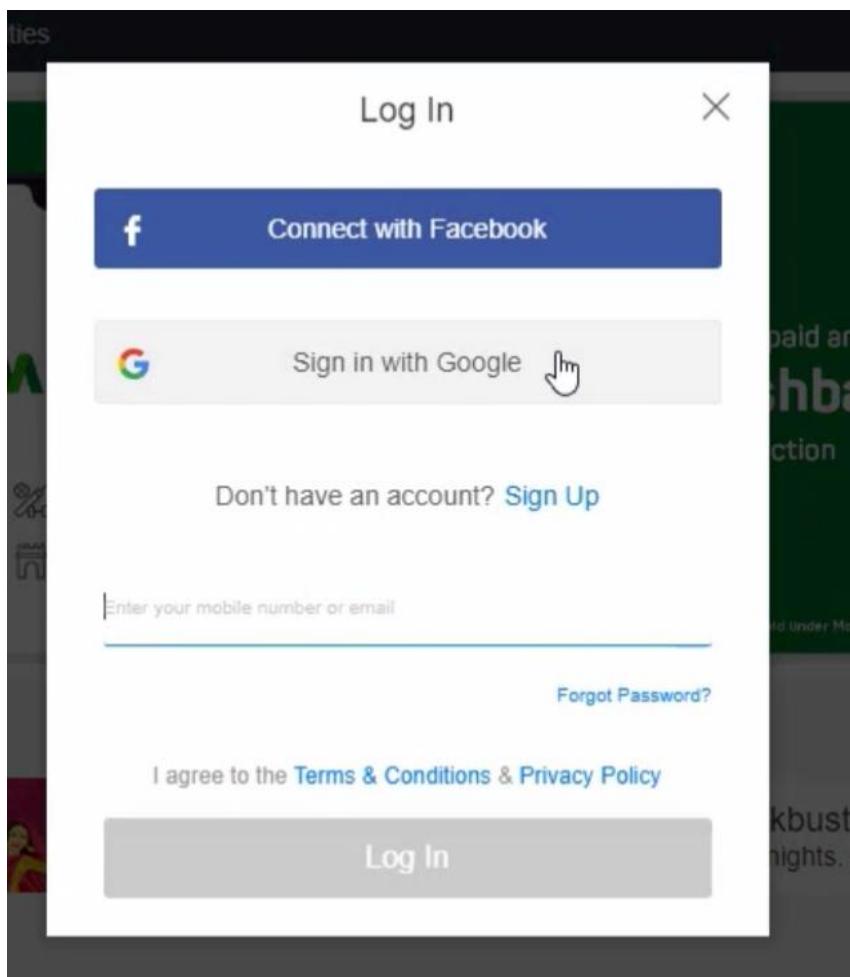
```
1  
2   "msg": "Delete Book operation failed, looks like the book doesnt exists"  
3 ]
```

Status: 404 Not Found Time: 835 ms Size: 495 B Save Response

Find and Replace Console Bootcamp Runner Train Udemy

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Automation Testing' selected, showing 'Collections' (with 'Library'), 'APIs', 'Environments', 'Mock Servers', 'Monitors', and 'History'. The main workspace is titled 'Library / DeleteBook' and contains a POST request to 'https://rahulshettyacademy.com/Library/DeleteBook.php'. The 'Body' tab is active, showing a JSON payload with an 'ID' field set to 'ytnft227'. The response status is '404 Not Found', with the message: 'The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.' Other tabs like 'Params', 'Authorization', 'Headers (9)', 'Tests', and 'Settings' are visible. At the bottom, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON'. A status bar at the bottom indicates 'Status: 404 Not Found', 'Time: 835 ms', and 'Size: 495 B'. The footer includes links for 'Bootcamp', 'Runner', 'Train', and 'Udemy'.

OAuth2.0



Why Applications rely on Other (Google or facebook) Authentications?

No Data breach Headaches for Application

Need not maintain user profile data

This also allows richer websites by allowing disparate applications to talk to each other.

User sign into Google by hitting google Authorization server and get code

Application will use this code to hit google resource server in back end to get Access token

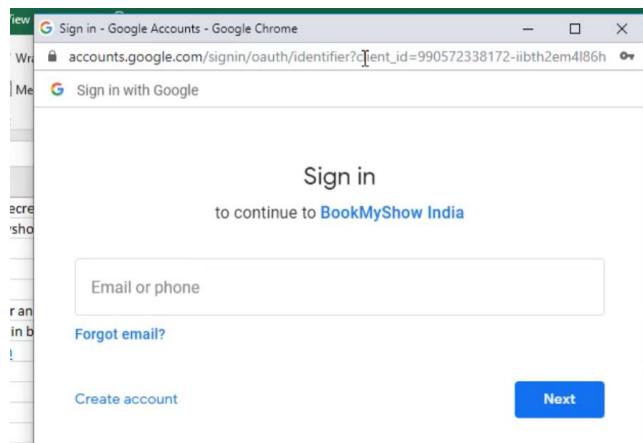
Application grants access to User by validating Access token

User sign into Google by hitting google Authorization server and get code

Application will use this code to hit google resource server in back end to get (Access token,First name, last name, email,d)

Application grants access to User by validating Access token

B	C	D	E	F
Client	ClientID	Client secret ID	Resource owner	Resource/Authorization Server
BookMyShow	ID that identifies the client	Bookmyshow registers this with google	Human	Google



The screenshot shows the Postman interface. On the left, there's a sidebar with collections like 'explore', 'oauth', and 'random'. The main area shows a request for 'actualrequest' with a GET method to https://rahulshettyacademy.com/getCourse.php?access_token=ya29.Glx1BwNn2k_FvPkQ9jj0SVvru... . The 'Params' tab is selected, showing a parameter 'access_token' with value ya29.Glx1BwNn2k_FvPkQ9jj0SVvru... . The response body shows the error message 'AUTHENTICATION FAILED !!! PLEASE ENTER VALID ACCESS TOKEN'.

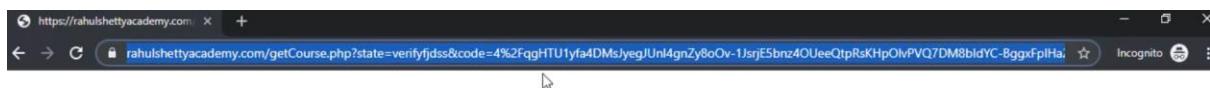
GET https://accounts.google.com/o/oauth2/v2/auth?
 Params ● Auth
 Query Params

KEY	VALUE	DESCRIPTION	***
<input checked="" type="checkbox"/> scope	https://www.googleapis.com/auth/userinfo.em...		
<input checked="" type="checkbox"/> auth_url	https://accounts.google.com/o/oauth2/v2/auth		
<input checked="" type="checkbox"/> client_id	692183103107-p0m7ent2hk7suguv4vq22hjcfhc...		
<input checked="" type="checkbox"/> response_type	code		
<input checked="" type="checkbox"/> redirect_uri	https://rahulshettyacademy.com/getCourse.php		
<input checked="" type="checkbox"/> state	verifyfdss		
Key	Value	Description	

Scope what information is required from google

State is optional it should be available in Rahul server for security purposes

After authorisation:



Copy url and paste and it formatted like below, in state its giving whats we given and what we received both are same

GET https://rahulshettyacademy.com/getCourse.php?state=verifyfdss&code=4%2FqgHTU1yfa4DMslye...
 Params ● Authorization Headers Body Pre-request Script Tests Cookies Code Com
 Query Params

KEY	VALUE	DESCRIPTION	***
<input checked="" type="checkbox"/> state	verifyfdss		
<input checked="" type="checkbox"/> code	4%2FqgHTU1yfa4DMslyegJUnl4gnZy8oOv-1jsrjE5bnz4OUeeQtpRsKHpOlPVQ7DM8bldYC-BggxFplHa2P2oL90g		
<input checked="" type="checkbox"/> scope			
<input checked="" type="checkbox"/> authuser	0		
<input checked="" type="checkbox"/> session_state	815b019bce702ca0a27eaeca6f308752ad5012d...		
<input checked="" type="checkbox"/> prompt	consent		
Key	Value	Description	

POST [https://www.googleapis.com/oauth2/v4/token?](https://www.googleapis.com/oauth2/v4/token?code=4%2FqQFb7KHEsY68tzTVl_pTnc4a_QufPB_qxC5InCBYcFnYpbcyT_OyvBPCVHBZmWoVsB84n0y mAkdKho2cP-kWprs&client_id=692183103107-p0m7ent2hk7suguv4vq22hjcfhc43pj.apps.googleusercontent.com&client_secret=erZOWM9g3UtwNR340YYak_W&redirect_uri=https://rahulshettyacademy.com/getCourse.php&grant_type=authorization_code)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> code	4%2FqQFb7KHEsY68tzTVl_pTnc4a_QufPB_qxC5...	
<input checked="" type="checkbox"/> client_id	692183103107-p0m7ent2hk7suguv4vq22hjcfhc...	
<input checked="" type="checkbox"/> client_secret	erZOWM9g3UtwNR340YYak_W	

GET <https://accounts.google.com/o/oauth2/v2/auth?scope=https://www.googleapis.com/auth/userinfo...>

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Cookies	Code
Query Params							
<input checked="" type="checkbox"/> scope							
<input checked="" type="checkbox"/> auth_url							
<input checked="" type="checkbox"/> client_id							
<input checked="" type="checkbox"/> response_type							
<input checked="" type="checkbox"/> redirect_uri							
<input checked="" type="checkbox"/> state							

GET <https://accounts.google.com/o/oauth2/v2/auth?scope=https://www.googleapis.com/auth/userinfo...>

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Cookies	Code	Comments (0)
<input checked="" type="checkbox"/> scope								
<input checked="" type="checkbox"/> auth_url								
<input checked="" type="checkbox"/> client_id								
<input checked="" type="checkbox"/> response_type								
<input checked="" type="checkbox"/> redirect_uri								
<input checked="" type="checkbox"/> state								
Key	Value	Description	***	Bulk Edit				
Key	Value	Description	***	Bulk Edit				

exchangeCode

POST https://www.googleapis.com/oauth2/v4/token?code=4%2FqQFb7KHEsY68zTV_l_pTnc4a_QufPB_qx... Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Cookies Code Comments

Query Params

KEY	VALUE	DESCRIPTION
code	4%2FqQFb7KHEsY68zTV_l_pTnc4a_QufPB_qxC5...	
client_id	692183103107-p0m7ent2hk7suguv4vg22hjcfhc...	
client_secret	erZoWM9g3UtwNRj340YYaK_W	
redirect_uri	https://rahulshettyacademy.com/getCourse.php	
grant_type	authorization_code	
Key	Value	Description

Body Cookies Headers (13) Test Results Status: 200 OK Time: 278ms Size: 1.56 KB Save Response

History Collections APIs BETA

New Collection Trash

explore 4 requests

getCode exchangecode actualrequest random

oauth 3 requests

oauthdemo https://rahulshettyacademy.com/getCourse.php c1

exchangeCode

POST https://www.googleapis.com/oauth2/v4/token?code=4%2FqQFb7KHEsY68zTV_l_pTnc4a_QufPB_qx... Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Cookies Code Comments

Query Params

KEY	VALUE	DESCRIPTION
client_secret	erZoWM9g3UtwNRj340YYaK_W	
redirect_uri	https://rahulshettyacademy.com/getCourse.php	
grant_type	authorization_code	
Key	Value	Description

Body Cookies Headers (13) Test Results Status: 200 OK Time: 1235ms Size: 1.56 KB Save Response

Pretty Raw Preview JSON

```

1 {
2   "access_token": "ya29.GIx1B3WXOeEjzJct5BKy81uoRpGZ1_0WS...",
3   "expires_in": 3577,
4   "scope": "https://www.googleapis.com/auth/userinfo.email openid",
5   "token_type": "Bearer",
6   "id_token": "eyJhbGciOiJSUzI1NiIwImtpZC16IMw22jgwZjM3ZIxYzIzITYvZjI2TQyHf1HjkY5ZDV20TusjKjLC30eXAiD1kV1QifQeyJpC3M105JodhRwczovL2FjY2910nRzImdBb3d2S5jb20lLCJhenI0I120TixODh0I0v0Oct0BtHv0u03paadzhd1dR2cTyaapjJmhjcjQzcdouX8Bwc5nb29n0v12cV0y29ud0vUc5j620lLCJzdwI0I1xH0Q2hzWfTiYfUz2130tg3NzciLC1bwFpbC18imNmuaF02X0NvYTZA2Z1halwuV29t1w12n1a0kfamwvazDpQ10nrydws1mP0X2hnc2g10123963RUYwicVhyN1BnZhKgH1ZOURK11w1anP01joNvT3Yj4R4lgSLC1Hm10jE1NjcyOD1z0019,
7 Ha2cKH_2qgeCRm-3c1c1H3xvIPx2yuyx_30JhNz2MkPhFOXPJtfK3hNeVaScVe-FigorTD-vKpRmPsyznPeTCyv00M46xQ1gzbKRt0dRrTFTgkBQoc3npFvgkhoopJ-1S9vt17x5z79mgcvSU83FSCE_s9cLnxu04X1LbtbG2sDvY2kXbr1ct_xj0kvAa9ASy5xcCivuBnQc7x7kk00X2b0wy_Rh80anCuYcPHG7LvrgrfUDEXH1IcgD3pAh1Hks5uffGRg8do86MS04JRj0hvJNdnLzlvtF_gk1-f_ZLQAXQTDeySwyM2RndkXp4Q"

```

actualrequest

GET https://rahulshettyacademy.com/getCourse.php?access_token=ya29.GIx1B3WXOeEjzJct5BKy81uoRpGZ1_0WS... Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Cookies Code Comments

Query Params

KEY	VALUE	DESCRIPTION
access_token	ya29.GIx1B3WXOeEjzJct5BKy81uoRpGZ1_0WS...	
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 1169ms Size: 561 B Save Response

Pretty Raw Preview HTML

```

1 {
2   "Instructor": "RahulShetty",
3   "url": "rahulshettyacademy.com",
4   "services": [
5     "WebAutomation", [
6       {"courseTitle": "Selenium Webdriver Java", "price": "50"}, ...
7     ],
8     {"courseTitle": "Cypress", "price": "40"}, ...
9     {"courseTitle": "Protractor", "price": "40"}, ...
10    {"API": [
11      {"courseTitle": "Rest Assured Automation using Java", "price": "50"}, ...
12      {"courseTitle": "SoapUI Webservices testing", "price": "40"}, ...
13    ],
14    "Mobile": [
15      {"courseTitle": "Appium-Mobile Automation using Java", "price": "50"} ...
16    ]
17  ]
18 }

```

Type here to search

```

1  {
2   "Instructor": "RanuiShetty",
3   "url": "rahulshettyacademy.com",
4   "services": "projectSupport",
5   "expertise": "Automation",
6   "courses": [
7     "WebAutomation": [
8       {
9         "courseTitle": "Selenium Webdriver Java",
10        "price": "50"
11      },
12      {
13        "courseTitle": "Cypress",
14        "price": "40"
15      },
16      {
17        "courseTitle": "Protractor",
18        "price": "40"
19      }
20    ],
21    "API": [
22      {
23        "courseTitle": "Rest Assured Automation using Java",
24        "price": "50"
25      },
26      {
27        ...
28      }
29    ]
30  }

```

OAuth 2.0 Contract Details:

GrantType	Authorization code
redirect URL/Callback URL	https://rahulshettyacademy.com/getCourse.php
Authorization server url	https://accounts.google.com/o/oauth2/v2/auth
Access token url	https://www.googleapis.com/oauth2/v4/token
Client ID	692183103107-p0m7ent2hk7suguv4vq22hjcfhcr43pj.apps.googleusercontent.com
Client Secret	erZOWM9g3UtwNRj340YYaK_W
Scope	https://www.googleapis.com/auth/userinfo.email
State	Any random string
How to pass oauth in request	Headers

Mandatory fields for GetAuthorization Code Request ;

End Point : Authorization server url

Query Params:Scope, Auth_url, client_id, response_type, redirect_uri

output : Code

Mandatory fields for GetAccessToken Request :

End point : Access token url

Query Params :Code, client_id, client_secret, redirect_uri, grant_type

Output : Access token

Python Automation:

```
1 import json
2
3 courses = '{"name": "RahulShetty", "languages": [ "Java", "Python"]}'
4
5 #Loads method parse json string and it returns dictionary
6 dict_courses = json.loads(courses)
7 print(type(dict_courses))
8 print(dict_courses)
9 print(dict_courses['name'])
10 #get me the first language taught by trainer
11 # list_language = dict_courses['languages']
12 # print(type(list_language))
13 # print(list_language[0])
14 print(dict_courses['languages'][0])
15
```

```
16 ##### Parse content present in Json file
17 with open('C:\\Users\\Owner\\Documents\\course.json') as f:
18     data = json.load(f)
19     print(data)
20     print(type(data))
21
```

Run: jsonParsing
RahulShetty
Java
{'dashboard': {'purchaseAmount': 910, 'website': 'rahulshettyacademy.com'}, 'courses': [{'title': 'Selenium Python', 'price': 45}, {"title": "RPA", "price": 45}, {"title": "API Testing", "price": 45}, {"title": "Machine Learning", "price": 45}], "type": "dict"}
I

```
for course in data['courses']:
    #print(course)
    if course['title'] == "RPA":
        print(course['price'])
        assert course['price'] == 45
```

```
32 with open('C:\\Users\\Owner\\Documents\\course1.json') as fi:
33     data2 = json.load(fi)
34     assert data == data2
35
```

requests 2.7.0

```
pip install requests==2.7.0
```

Get book by author name

The screenshot shows the Postman interface. On the left, there's a sidebar with collections like 'Collections', 'Jira', and 'Library API'. The main area shows a GET request to 'http://216.10.245.166/Library/GetBook.php?AuthorName=Rahul Shetty2'. In the 'Params' tab, 'AuthorName' is set to 'Rahul Shetty2'. The response body is displayed as JSON:

```
1  {
2   "book_name": "Python Selenium 18hrs By Rahul Shetty",
3   "isbn": "PSRS",
4   "aisle": "19"
5 }
```

```
import requests

requests.get(
```

A tooltip for the 'get' method is open, showing suggestions: 'get(url, params, kwargs)', 'requests.api', and 'requests.packages'. A note at the bottom says: 'Press Ctrl+ to choose the selected (or first) suggestion and insert a dot afterwards. Next Tip'.

In get method url and parameters are separated by ?

```
GET http://216.10.245.166/Library/GetBook.php?AuthorName=Rahul Shetty2
```

```
import requests
import json

response = requests.get('http://216.10.245.166/Library/GetBook.php',
                       params={'AuthorName': 'Rahul Shetty2'})
print(response.text)
print(type(response.text))
dict_response = json.loads(response.text)
print(type(dict_response))
```

Run: apiValidations

```
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmProjects1/BackEndAutomation/apiValidations.py
[{"book_name": "Python Selenium 18hrs By Rahul Shetty", "isbn": "PSRS", "aisle": "19"}]
<class 'str'>
<class 'list'>

Process finished with exit code 0
```

```
10 json_response = response.json()
11 print(type(json_response))
```

Run: apiValidations

```
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmProjects1/BackEndAutomation/apiValidations.py
<class 'list'>
```

Headers in postman

The screenshot shows the 'Headers' tab in Postman with 10 entries. The headers listed are:

KEY	VALUE
Date	Fri, 05 Jun 2020 22:12:53 GMT
Server	Apache
Access-Control-Allow-Origin	*
Access-Control-Allow-Methods	POST
Access-Control-Max-Age	3600
Access-Control-Allow-Headers	Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With
Keep-Alive	timeout=5, max=100
Connection	Keep-Alive
Transfer-Encoding	chunked
Content-Type	application/json;charset=UTF-8

The screenshot shows the 'Tests' tab in Postman with the following Python script:

```
13 | assert response.status_code == 200
14 | print(response.headers)
15 |
16 |
17 |
18 |
```

The output of the test is displayed below:

```
Connection': 'Keep-Alive', 'Transfer-Encoding': 'chunked', 'Content-Type': 'application/json;charset=UTF-8'
```

Sometimes we need to verify that what is the content type we received

Getting book by author name and check isbn of the book and if exist retrieve the book details

The screenshot shows a GET request to <http://216.10.245.166/Library/GetBook.php?AuthorName=Rahul Shetty>. The 'Params' tab is selected, showing a single parameter 'AuthorName' with value 'Rahul Shetty'. The 'Body' tab shows the JSON response:

```
[{"book_name": "Learn API Automation with Java", "isbn": "xxx", "aisle": "100"}, {"book_name": "Learn Java", "isbn": "JDBCIN", "aisle": "231"}]
```

```
json_response = response.json()
```

it gives list of dict.

```

ot
py 16     # Retrieve the book details with ISBN RGHCC
17     for actualBook in json_response:
18         if actualBook['isbn'] == 'RGHCC':
19             print(actualBook)
20             break
21
22     expectedBook = {
23         "book_name": "Learn API Automation with RestAssured",
24         "isbn": "RGHCC",
25         "aisle": "12239"
26     }
27
28     assert actualBook == expectedBook

```

ins ×

```

k_name': 'Learn API Automation with RestAssured', 'isbn': 'RGHCC', 'aisle': '12239'}
```

```

ss finished with exit code 0
```

POST API

Library API :

BaseURI : http://216.10.245.166

1. **Resource** : Library/AddBook.php **Method** : POST

Input Payload : Json:

```
{
  "name": "Learn Appium Automation with Java",
  "isbn": "bcd",
  "aisle": "227",
  "author": "John doe"
}
```

Output Json

```
{
  "Msg": "successfully added",
  "ID": "bcd227"
}
```

The screenshot shows a code editor with several tabs at the top: 'nParsing.py', 'apiValidations.py', and 'postAPIExample.py'. The 'postAPIExample.py' tab is active. The code in the editor is:

```

import requests

requests.post
    f post(url, data, json, kwar...  requests.api
    Ctrl+Down and Ctrl+Up will move caret down and up in the editor  Next Tip

```

POST http://216.10.245.166/Library/Addbook.php

Headers (10)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Content-Type	application/json	
Key	Value	Description

Status: 200 OK Time: 717 ms Size: 459 B Save

```

import requests
addBook_response = requests.post('http://216.10.245.166/Library/Addbook.php', json={
    "name": "Learn Appium Automation with Java",
    "isbn": "bdcdb",
    "aisle": "22fd7",
    "author": "John doe"
}, headers={"Content-Type": "application/json"})
print(addBook_response.json())

```

C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmProjects1/BackEnd
{'Msg': 'successfully added', 'ID': 'bdcdb22fd7'}
Process finished with exit code 0

```

print(addBook_response.json())
response_json = addBook_response.json()
print(type(response_json))

bookId = response_json['ID']

```

1. Resource :/Library/DeleteBook.php Method : POST

Input Payload : Json:

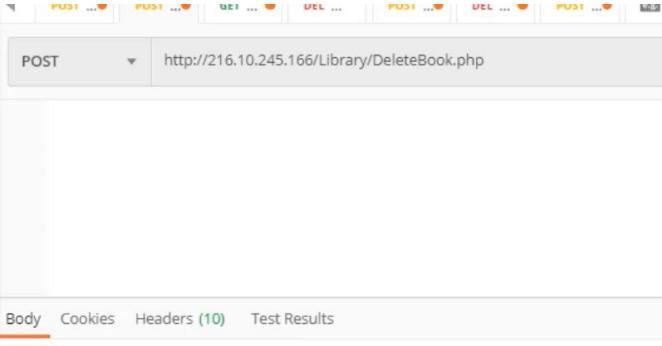
{

"ID": "a23h345122332"

}

Output Response :

Using the POST method for deleting data is not a recommended practice according to the HTTP specification. The DELETE method is specifically designed for deletion operations in RESTful APIs. However, there might be cases where the DELETE method is not allowed due to various reasons such as restrictions imposed by some web servers or frameworks, or limitations in certain environments. In such cases, developers might resort to using the POST method with a designated endpoint or parameter to perform deletion operations. However, it's important to note that this deviates from standard RESTful API conventions and may not be the best practice.



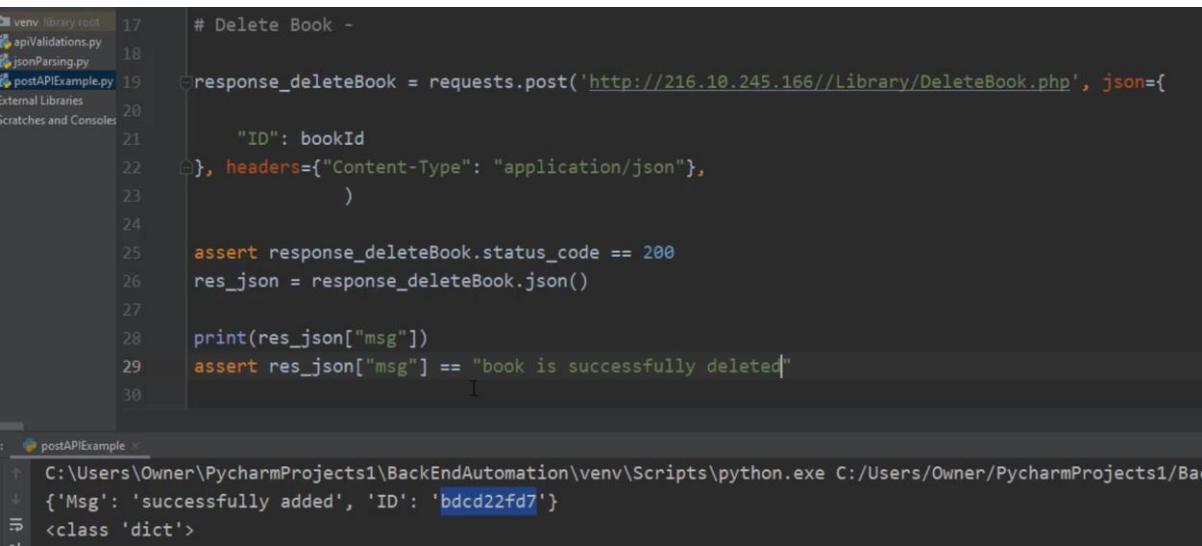
Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2   "msg": "book is successfully deleted"
3 }

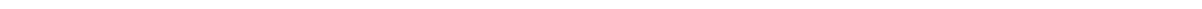
```



```

17 # Delete Book -
18
19 response_deleteBook = requests.post('http://216.10.245.166//Library/DeleteBook.php', json={
20
21     "ID": bookId
22 }, headers={"Content-Type": "application/json"},
23 )
24
25 assert response_deleteBook.status_code == 200
26 res_json = response_deleteBook.json()
27
28 print(res_json["msg"])
29 assert res_json["msg"] == "book is successfully deleted"
30

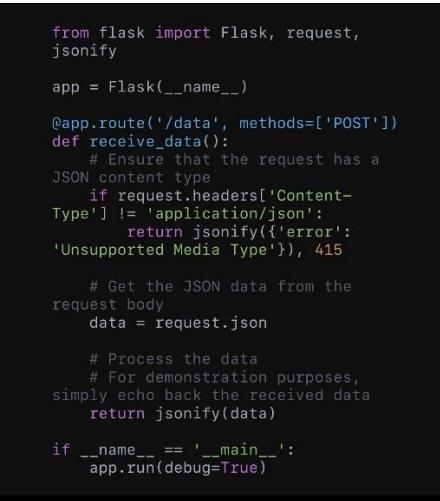
```



```

C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmProjects1/Ba
{'Msg': 'successfully added', 'ID': 'bdc22fd7'}
<class 'dict'>

```



```

from flask import Flask, request,
jsonify

app = Flask(__name__)

@app.route('/data', methods=['POST'])
def receive_data():
    # Ensure that the request has a
    # JSON content type
    if request.headers['Content-
    Type'] != 'application/json':
        return jsonify({'error':
        'Unsupported Media Type'}), 415

    # Get the JSON data from the
    # request body
    data = request.json

    # Process the data
    # For demonstration purposes,
    # simply echo back the received data
    return jsonify(data)

if __name__ == '__main__':
    app.run(debug=True)

```

In headers we are sending content type based on content type which received from api backend code where designed so for integration process we used content type

Creating Global Variables:

The screenshots show the development process of creating global variables for an API endpoint in a Python project named "BackEndAutomation".

- properties.ini:** Contains the configuration section [API] with the endpoint set to `http://216.10.245.166`.
- postAPIExample.py:** Imports json, configparser, and payLoad. It uses the endpoint from properties.ini to make a POST request to `/Library/Addbook.php` with a payload of `"fefrewef"` and headers `{"Content-Type": "application/json"}`.
- configurations.py:** Imports configparser. It defines a function `getConfig()` which reads the properties.ini file and returns the configuration object.
- resources.py:** Imports json, configparser, payLoad, and configurations. It defines a class `ApiResources` with methods for adding, deleting, and getting book authors using the endpoint from the configuration object.

```
[API]
endpoint = http://216.10.245.166

import json
import configparser
from payLoad import *

import requests
config = configparser.ConfigParser()
config.read('utilities/properties.ini')
addBook_response = requests.post(config['API']['endpoint']+ '/Library/Addbook.php',
                                  json=addBookPayload("fefrewef"),
                                  headers={"Content-Type": "application/json"}, )

import configparser

def getConfig():
    config = configparser.ConfigParser()
    config.read('utilities/properties.ini')
    return config

import json
import configparser
from payLoad import *
from utilities.configurations import *

import requests
addBook_response = requests.post(getConfig()['API']['endpoint']+ '/Library/Addbook.php',
                                  json=addBookPayload("fefrewef"),
                                  headers={"Content-Type": "application/json"}, )

class ApiResources:
    addBook = '/Library/Addbook.php'
    deleteBook = '/Library/DeleteBook.php'
    getBookAuthor = ''
```

```

BackEndAutomation C:\Users\...
Project  Pr.  ⌂  ⌂  ⌂  -  jsonParsing.py  apiValidations.py  postAPIExample.py  resources.py  payLoad.py  configurations.py  properties.ini
  BackEndAutomation C:\Users\...
    | utilities
    |   __init__.py
    |   configurations.py
    |   properties.ini
    + resources.py
      + venv library root
        + apiValidations.py
        + jsonParsing.py
        + payLoad.py
        + postAPIExample.py
      + External Libraries
      + Scratches and Consoles
postAPIExample.py
from utilities.resources import *
from utilities.configurations import *

import requests

url = getConfig()['API']['endpoint'] + ApiResources.addBook
headers = {"Content-Type": "application/json"}
addBook_response = requests.post(url,json=addBookPayload("fefrewewe"),headers=headers, )
print(addBook_response.json())

```

```

Load.py
postAPIExample.py
External Libraries
Scratches and Consoles
postAPIExample x
31     #Authentication
32     url = "https://api.github.com/user"
33     github_response = requests.get(url,auth=('rahulshettyacademy',getPassword()))
34
35     print(github_response.status_code)
36

```

```

postAPIExample x
fefrewewe227
book is successfully deleted
200

```

```

BackEndAutomation C:\Users\...
Project  Pr.  ⌂  ⌂  ⌂  -  jsonParsing.py  apiValidations.py  postAPIExample.py  resources.py  payLoad.py  configurations.py  properties.ini
  BackEndAutomation C:\Users\...
    | utilities
    |   __init__.py
    |   configurations.py
    |   properties.ini
    + resources.py
      + venv library root
        + apiValidations.py
        + jsonParsing.py
        + payLoad.py
        + postAPIExample.py
      + External Libraries
      + Scratches and Consoles
postAPIExample x
33     url = "https://api.github.com/user"
34     github_response = requests.get(url,verify=False,auth=('rahulshettyacademy',getPassword()))
35
36     print(github_response.status_code)
37
38     url2 = "https://api.github.com/user/repos"
39     response= requests.get(url2,auth=('rahulshettyacademy',getPassword()))
40     print(response.status_code)
41
42
43
44

```

```

postAPIExample x
InsecureRequestWarning,
200
200

```

if we want to access multiple apis having auth we have to pass auth in every request or else we can create session

```

batchFiles
outputFiles
utilities
venv library root
  ad.py
  apivValidations.py
  csvDemo.py
  dbDemo.py
  jsonParsing.py
  payLoad.py
  postAPIExample.py
  sshConnectDemo.py
  webScraping.py
  webScraping2.py
External Libraries
Scratches and Consoles

#Authentication
se = requests.session()
se.auth = auth=('rahulshettyacademy', getPassword())

url = "https://api.github.com/user"
github_response = requests.get(url, verify=False, auth=('rahulshettyacademy', getPassword()))

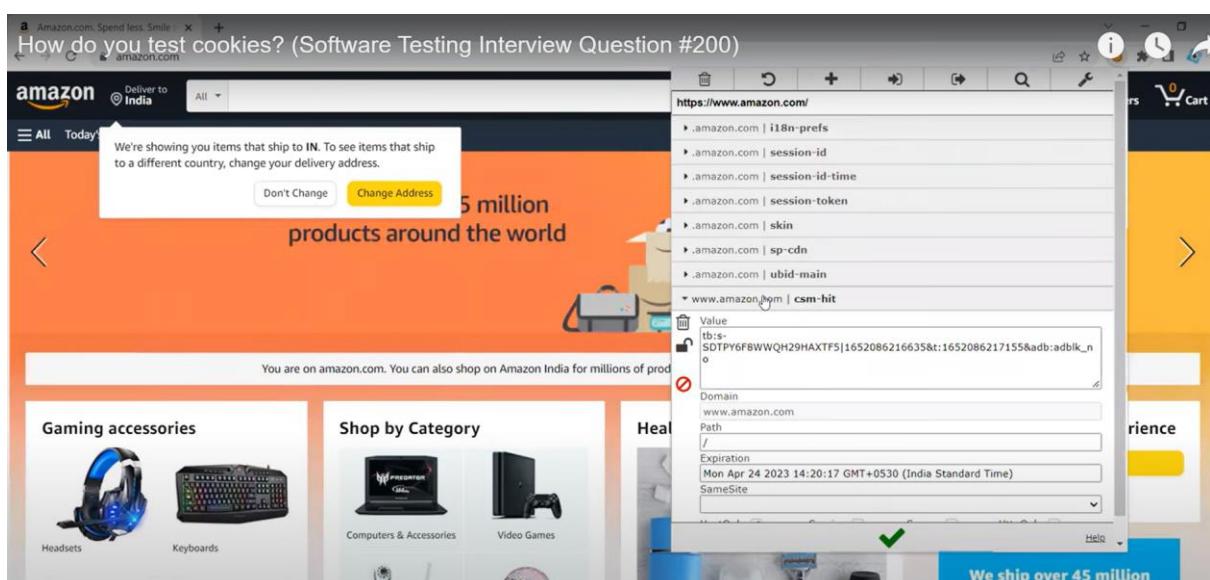
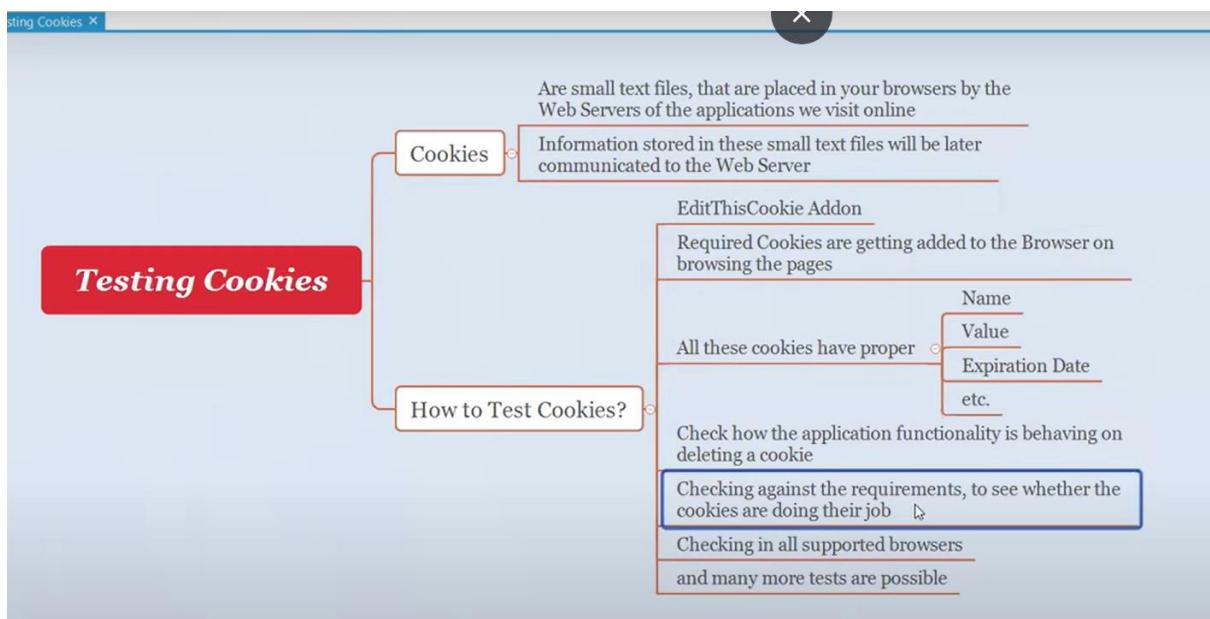
print(github_response.status_code)

url2 = "https://api.github.com/user/repos"
response= se.get(url2)
print(response.status_code)

Run: postAPIExample
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmProjects1/BackEndAutomation/postAPIExample.py
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\lib\site-packages\urllib3\connectionpool.py:986: InsecureRequestWarning: 
  InsecureRequestWarning,
  200
  200

```

Cookies





There is one my website with Http.

Rahul Shetty.

Academy.com.

Right.

So my website have one cookie.

So basically when people land, we will see what is the visit month that they visited last.

Okay.

So if their visit month is, let's say this is August and if they have visited this in June, so if

there are no site changes after June, we will load the website from the cache, what they already have

so that response time will be fast.

Okay.

And if you have updated our website after June and if the visit month is June, so then we will bring

this website from the real server without caching.

So that's where we use cache in our website and.

To track what is the visit month?

We use cookies, so whenever people land on web page, we use a cookie called Visit month.

So this is a common like every website have a number of cookies to give best experience to the users.

So whenever they land, we use a cookie called visit month.

So we will read this cookie.

And based upon that, as I told, we will load our web page.

The screenshot shows the PyCharm IDE interface with three code editors open:

- Top Editor:** Displays a script named `Miscellaneous.py` containing code to send a GET request to `http://rahulshettyacademy.com` with a cookie for the month of February. It prints the status code.
- Middle Editor:** Displays a script named `Miscellaneous.py` containing code to send a GET request to `https://httpbin.org/cookies` with a cookie for the year 2022. It prints the response text, which shows a JSON object with a "cookies" key containing a "visit-year" value of "2022".
- Bottom Editor:** Displays a script named `Miscellaneous.py` containing code to use a session to update the cookie for the month of February and then send a GET request to `https://httpbin.org/cookies` with a cookie for the year 2022. It prints the response text, showing a JSON object with a "cookies" key containing both "visit-month" and "visit-year" values.

If any redirection present in url it gives 301 response code for security reasons
 redirects present , like all the security things mentioned in one url first it will hit that n
 then it redirects to other url, but if developers added all security to other also we
 need to make sure that url is not redirecting by checking response history or one
 parameter allow_redirects its bool

The screenshot shows a PyCharm project structure for 'BackEndAutomation' with files like 'batchFiles.py', 'outputFiles.py', 'utilities.py', and various 'Demo' and 'Validation' scripts. The 'Miscellaneous.py' file is selected in the editor. The code uses the 'requests' library to make a GET request to 'http://rahulshettyacademy.com'. It includes a cookie for the month ('visit-month') and prints the response history and status code. The terminal output shows the script running and printing the status code 301.

```
import requests
#http://rahulshettyacademy.com
#'visit-month'
cookie = {'visit-month': 'February'}
response = requests.get('http://rahulshettyacademy.com', cookies=cookie)
#301,200
print(response.history)
print(response.status_code)

se = requests.session()
```

```
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmPro
[<Response [301]>]
200
```

Below its not allowing any redirection hence status code 301

The screenshot shows the same 'Miscellaneous.py' file with a modification: the 'allow_redirects=False' parameter is added to the 'requests.get' call. This prevents the script from following redirects, resulting in a status code of 301.

```
response = requests.get('http://rahulshettyacademy.com', allow_redirects=False, cookies=cookie)
#301,200
#print(response.history)
print(response.status_code)

se = requests.session()
```

```
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmProjects1/BackEndA
301
```

Timeouts

The screenshot shows the addition of a 'timeout=1' parameter to the 'requests.get' call. This limits the execution time of the request to 1 second, demonstrating how to handle timeouts in network operations.

```
visitmonth : February
requests.get('http://rahulshettyacademy.com', allow_redirects=False, cookies=cookie, timeout=1)
```

POST a Multipart-Encoded File

Requests makes it simple to upload Multipart-encoded files:

```
>>> url = 'https://httpbin.org/post'
>>> files = {'file': open('report.xls', 'rb')}
      ^
>>> r = requests.post(url, files=files)
>>> r.text
{
...
"files": {
    "file": "<censored...binary...data>"
},
...
}
```

You can set the filename, content_type and headers explicitly:

```
>>> url = 'https://httpbin.org/post'
>>> files = {'file': ('report.xls', open('report.xls', 'rb'), 'application/vnd.ms-exc
      ^
>>> r = requests.post(url, files=files)
>>> r.text
{
```

40. Sending Attachments through Post request call using Files

pet Everything about your Pets

POST /pet/{petId}/uploadImage uploads an image

Parameters

Name	Description
petId <small>* required</small> integer(\$int64) (path)	ID of pet to update petId - ID of pet to update
additionalMetadata string (formData)	Additional data to pass to server additionalMetadata - Additional data to pass !
file file (formData)	file to upload Choose File No file chosen

Responses

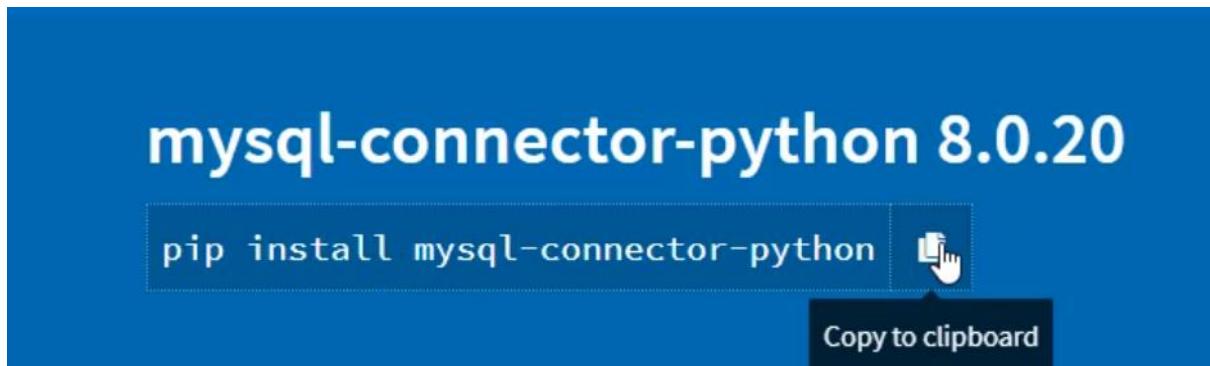
Code	Description
200	successful operation Example Value Model <pre>{ "code": 0, "type": "string", "message": "string" }</pre>

```
18 #Attachments
19 url = "https://petstore.swagger.io/v2/pet/9843217/uploadImage"
20 files = {'file': open('C:\\\\Users\\\\Owner\\\\Documents\\\\ra.png', 'rb')}
21 r = requests.post(url, files=files)
22 print(r.status_code)
23 print(r.text)
24

Run: Miscellaneous <
      }
    }

200
{
  "code":200,"type":"unknown","message":"additionalMetadata: null\\nFile uploaded to ./ra.png, 81685 bytes"
}

Process finished with exit code 0
```



To connect mysql server

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with various icons. Below it is a search bar labeled 'Find' and a 'Limit to 1000 rows' dropdown. The main area displays the following SQL code:

```

1 • CREATE DATABASE PythonAutomation;
2 use PythonAutomation;
3 CREATE TABLE CustomerInfo
4
5     (CourseName varchar(50),
6
7     PurchasedDate date,
8
9     Amount int(50),
10
11    Location varchar(50));
12

```

Below the code, there's a 'Result Grid' section with a table containing the following data:

	BookName	isbn	aisle	author
▶	Appium	fdsefr3	43	Rahul Shetty
	Devops	bnid34	75	Rahul Shetty2
	Selenium	kosncts34fr	23	Rahul Shetty
	Jmeter	rtbrss24t	234	Rahul Shetty3

The screenshot shows a PyCharm project named 'BackEndAutomation'. The file 'dbDemo.py' is open in the editor, displaying the following code:

```

import mysql.connector
#host, database,user,password
conn = mysql.connector.connect(host='localhost',database='PythonAutomation',
                                user='root',password='rahulshettyacademy')
print(conn.is_connected())

```

The project structure on the left shows files like postAPIExample.py, payLoad.py, apiValidations.py, resources.py, dbDemo.py, properties.ini, and utilities. The 'Run' tab at the bottom shows the output of the 'dbDemo' run configuration.

Above we established connection through conn

Below is cursor to send query and get result back

The screenshot shows the PyCharm IDE interface with three code editors and a run console.

- Top Editor:** Shows a MySQL connection script. A tooltip for the `fetchone` method is displayed, listing its signature and the class it belongs to (CMySQLCursor). The code includes a query to select all from CustomerInfo.
- Middle Editor:** Shows the same script with the `fetchone` call replaced by `row = cursor.fetchone()`. The run console below shows the output: `True` followed by a tuple: `('selenium', datetime.date(2020, 6, 7), 120, 'Africa')`.
- Bottom Editor:** Shows the script again with the `fetchone` call replaced by `rowAll = cursor.fetchall()`. The run console shows the output: `True` followed by a list of tuples: `['selenium', datetime.date(2020, 6, 7), 120, 'Africa']`.

fetchall returns list of tuples and it started where cursor was there earlier

CustomerInfo 6

Result Grid | Filter Rows: [] | Export: [] | Wrap Cell Content:

	CourseName	PurchasedDate	Amount	Location
▶	selenium	2020-06-07	120	Africa
	Protractor	2020-06-07	45	Africa
	Appium	2020-06-07	99	Asia
	WebServices	2020-06-07	21	Asia
	Jmeter	2020-06-07	76	Asia

CustomerInfo 6

Output ::::

```

14
15
16
17
18
19
20
21
22
23
24
25
rows = cursor.fetchall()
print(type(rows))
print(rows)
for row in rows: #('selenium', datetime.date(2020, 6, 7), 120, 'Africa')
    print(row[2])

conn.close()

```

Run: dbDemo

120
45
99
21
76

```

15
16
17
18
19
20
21
22
23
24
25
rows = cursor.fetchall()
print(type(rows))
print(rows)
sum = 0
for row in rows: #('selenium', datetime.date(2020, 6, 7), 120, 'Africa')
    sum = sum + row[2]

print(sum)

```

Run: dbDemo

120
45
361

```

query = "update customerInfo set Location = %s where CourseName = %s"
data = ("UK", "Jmeter")
cursor.execute(query,data)
conn.commit()

```

```
[API]
endpoint = http://216.10.245.166
[SQL]
user = localhost
password = rahulshettyacademy
database = PythonAutomation
host = localhost
```

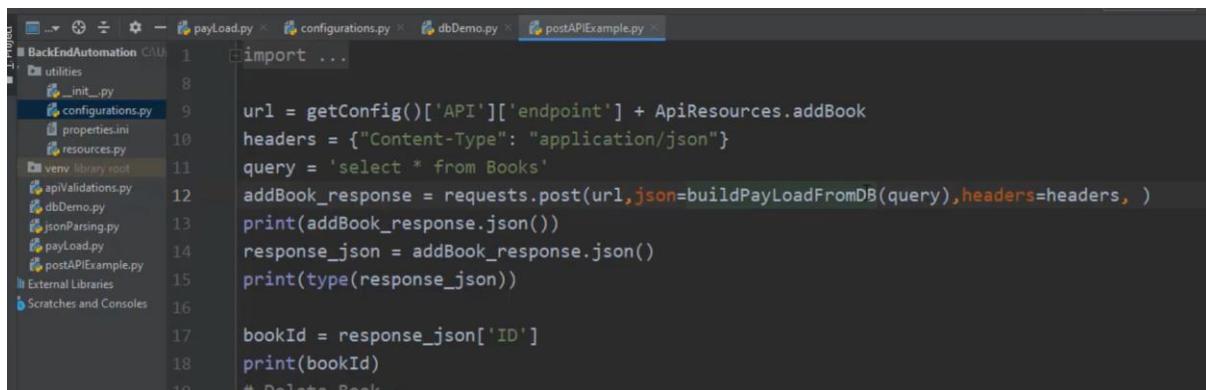
```
config.read('utilities/properties.ini')
return config

connect_config = {
    'user': getConfig()['SQL']['user'],
    'password': getConfig()['SQL']['password'],
    'host': getConfig()['SQL']['host'],
    'database': getConfig()['SQL']['database'],
}
```

```
return "gdfgdfgdfg"

def getConnection():
    try:
        conn = mysql.connector.connect(**connect_config)
        if conn.is_connected():
            print("Connection Successful")
            return conn
    except Error as e:
        print(e)
```

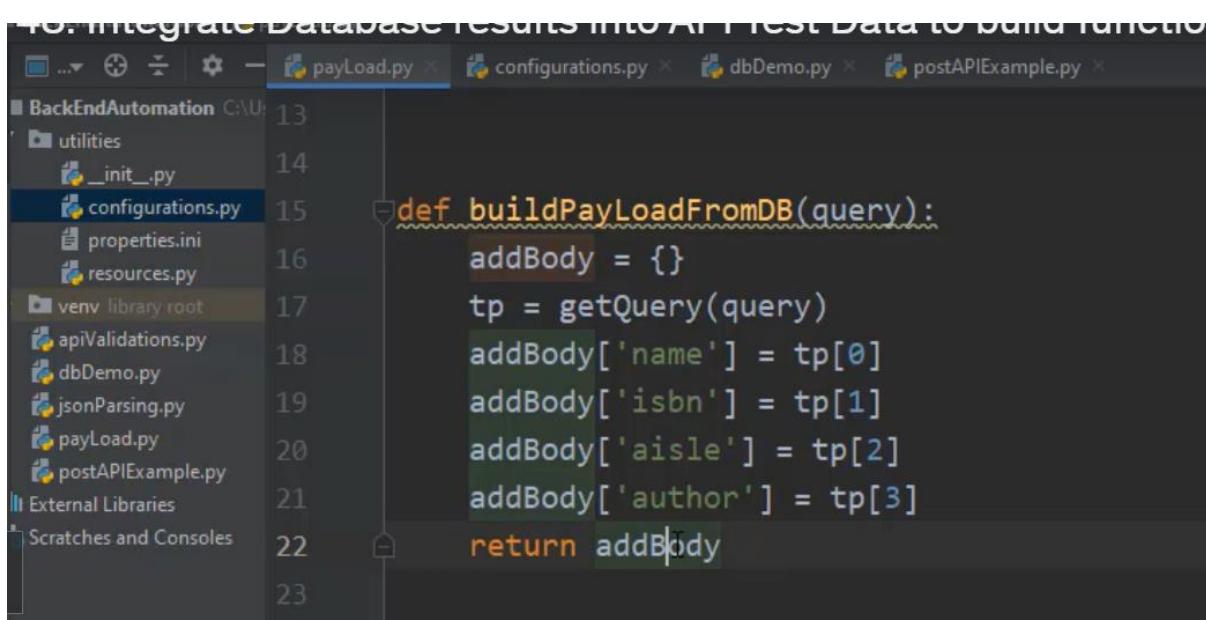
```
import mysql.connector
from utilities.configurations import *
#host, database, user, password
#conn = mysql.connector.connect(host='localhost', database='PythonAutomation',
#                                user='root', password='rahulshettyacademy')
conn = getConnection()
print(conn.is_connected())
cursor = conn.cursor()
cursor.execute('select * from CustomerInfo')
```



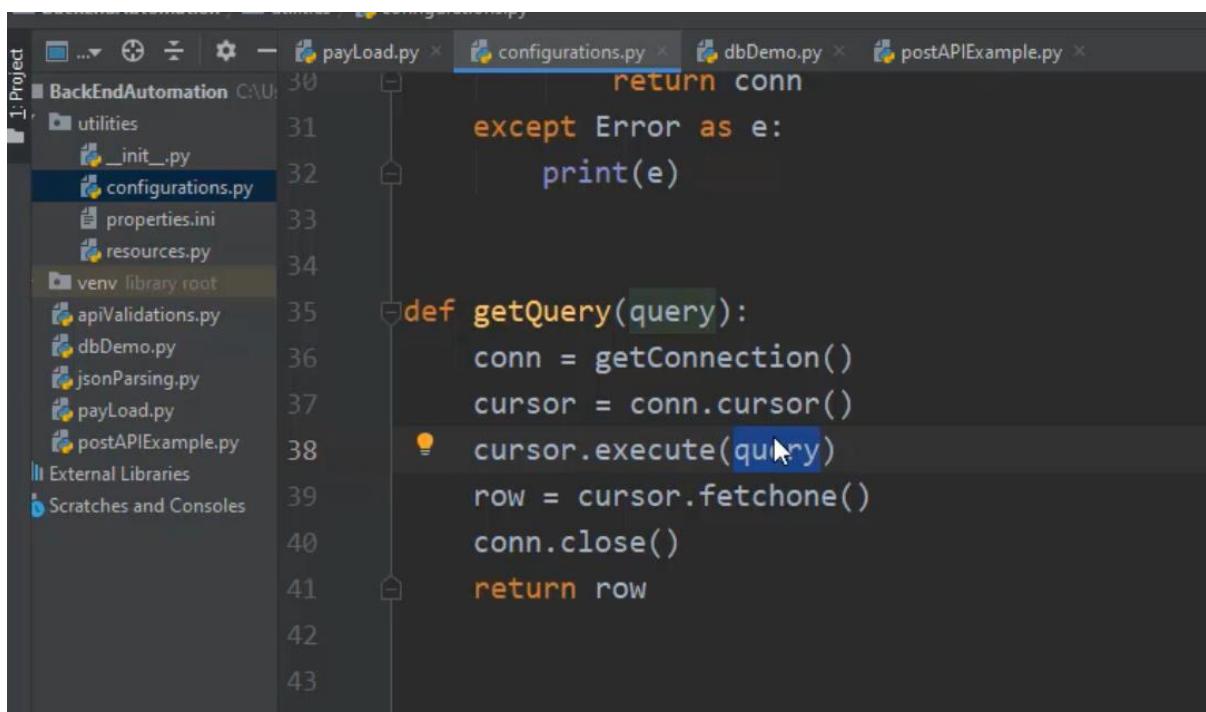
```
import ...

url = getConfig()['API']['endpoint'] + ApiResources.addBook
headers = {"Content-Type": "application/json"}
query = 'select * from Books'
addBook_response = requests.post(url,json=buildPayLoadFromDB(query),headers=headers, )
print(addBook_response.json())
response_json = addBook_response.json()
print(type(response_json))

bookId = response_json['ID']
print(bookId)
# Delete Book
```



```
def buildPayLoadFromDB(query):
    addBody = {}
    tp = getQuery(query)
    addBody['name'] = tp[0]
    addBody['isbn'] = tp[1]
    addBody['aisle'] = tp[2]
    addBody['author'] = tp[3]
    return addBody
```



```
return conn
except Error as e:
    print(e)

def getQuery(query):
    conn = getConnection()
    cursor = conn.cursor()
    cursor.execute(query)
    row = cursor.fetchone()
    conn.close()
    return row
```

Python SSH Connection to Linux:

Launching AWS Linux Instance

Update Configurations to accept SSH
Connection

Build Paramiko SSH Connection

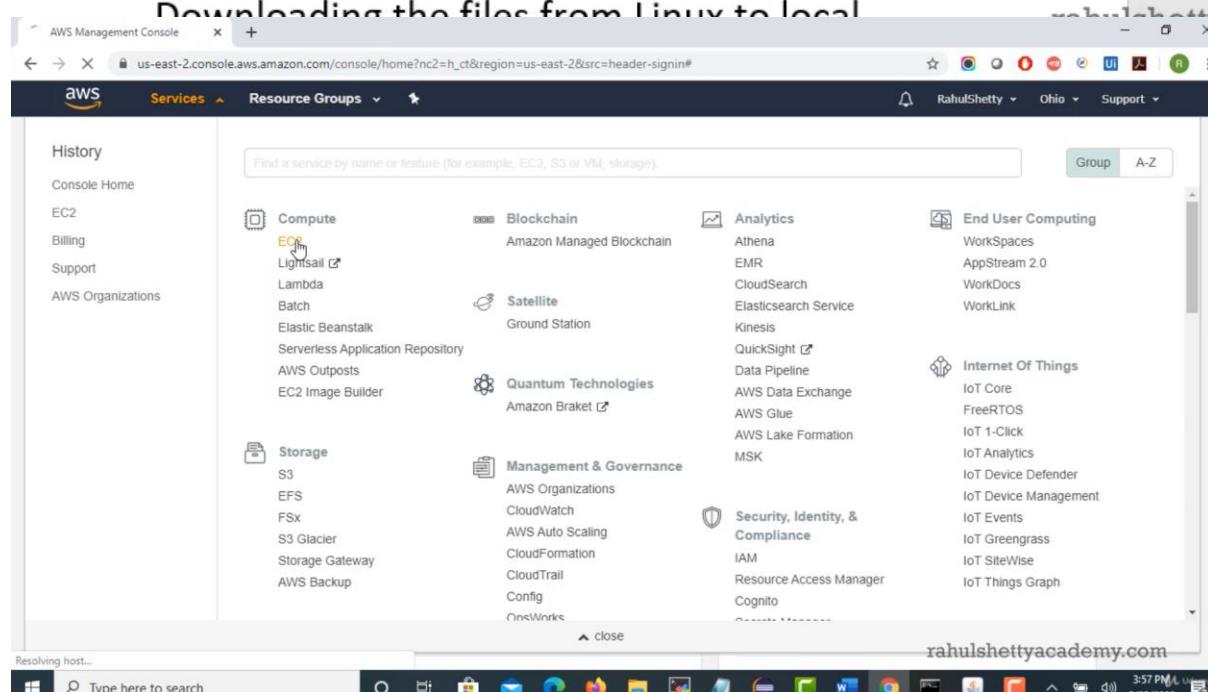
Run the commands from local Python Script on
Linux via SSH

How to read the output from Linux machine

How to transfer files from local server to Linux
Server

Running Python batch files in Linux server via
SSH

Downloading the files from Linux to local



Dashboard | EC2 Management C X + ↗ us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Home: RahulShetty Ohio Support

New EC2 Experience Tell us what you think

EC2 Dashboard New Events New Tags Reports Limits

INSTANCES Instances Instance Types Launch Templates Spot Requests Savings Plans Reserved Instances Dedicated Hosts New Capacity Reservations

IMAGES AMIs Bundle Tasks

Dedicated Hosts 0 Snapshots 0 Volumes 2 Load balancers 0 Key pairs 2 Security groups 4 Placement groups 0

Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using the AWS Launch Wizard for SQL Server. Learn more X

Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

Launch instance ▲ Launch instance from template US East (Ohio) Region

Scheduled events C

Default VPC vpc-dda874b6 Settings EBS encryption Zones Console experiments

Explore AWS Save up to 90% on EC2 with Spot Instances Optimize price-performance by combining EC2 purchase options in a single EC2 ASG. Learn more

Easily launch third-party AMI products AWS Marketplace has thousands of third-party AMI products that you can find, buy, and deploy with 1-click

Feedback English (US) Type here to search © 2008 - 2020, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use 4:03 PM

Launch instance wizard | EC2 Ma X + ↗ us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard: RahulShetty Ohio Support

Services Resource Groups

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 1: Choose an Amazon Machine Image (AMI)

Quick Start

My AMIs

Amazon Linux Free tier eligible

Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-0f7919c33c90f5b58 (64-bit x86) / ami-050d581a8c1d4a570 (64-bit Arm)
Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemv 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras.
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select 64-bit (x86) 64-bit (Arm)

Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-083ebc5a49573896a
The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select 64-bit (x86)

Red Hat Enterprise Linux 8 (HVM), SSD Volume Type - ami-0a54aef4ef3b5f881 (64-bit x86) / ami-0ffd59b53e6797671
Red Hat Enterprise Linux version 8 (HVM), EBS General Purpose (SSD) Volume Type
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select 64-bit (x86) 64-bit (Arm)

Feedback English (US) Type here to search © 2008 - 2020, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use 4:03 PM

Launch instance wizard | EC2 Ma x +

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 1: Choose an Amazon Machine Image (AMI)

Free tier eligible Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Microsoft Windows Server 2016 Base - ami-0a16ffe32a92704ea Select 64-bit (x86)
Windows Microsoft Windows 2016 Datacenter edition. [English]
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Microsoft Windows Server 2016 Base with Containers - ami-0e6af55522097f93a Select 64-bit (x86)
Windows Microsoft Windows 2016 Datacenter edition with Containers. [English]
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Deep Learning AMI (Microsoft Windows Server 2016) - ami-0389ad8f52eb83ff Select 64-bit (x86)
Windows Microsoft Windows Server 2016 with Tensorflow, Caffe and MXNet. [English]
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Microsoft Windows Server 2016 with SQL Server 2016 Standard - ami-08ca9fa2de549ef1d Select 64-bit (x86)
Windows Microsoft Windows 2016 Datacenter edition, Microsoft SQL Server 2016 Standard [English]
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Cancel and Exit

This screenshot shows the 'Choose an Amazon Machine Image (AMI)' step of the AWS Launch Instance Wizard. It lists several Windows-based AMIs, each with a 'Select' button and a 64-bit (x86) option. The first item, 'Microsoft Windows Server 2016 Base', is highlighted.

Feedback English (US)

Type here to search

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Search for an AMI by entering a search term e.g. "Windows"

Search by Systems Manager parameter

Quick Start

My AMIs

Amazon Linux Select 64-bit (x86)
Free tier eligible Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-0f7919c33c90f5b58 (64-bit x86) / ami-050d581a8c1d4a570 (64-bit Arm)
Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras.
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Community AMIs

Free tier only

Amazon Linux Select 64-bit (x86)
Free tier eligible Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-083ebc5a49573896a
The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Cancel and Exit

This screenshot shows the 'Choose an Amazon Machine Image (AMI)' step of the AWS Launch Instance Wizard. It lists two Linux-based AMIs: 'Amazon Linux 2 AMI (HVM)' and 'Amazon Linux AMI 2018.03.0 (HVM)'. Both have 'Select' buttons and 64-bit (x86) options. The first item, 'Amazon Linux 2 AMI (HVM)', is highlighted.

Feedback English (US)

Type here to search

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

This screenshot shows the 'Choose an Amazon Machine Image (AMI)' step of the AWS Launch Instance Wizard. It lists two Linux-based AMIs: 'Amazon Linux 2 AMI (HVM)' and 'Amazon Linux AMI 2018.03.0 (HVM)'. Both have 'Select' buttons and 64-bit (x86) options. The first item, 'Amazon Linux 2 AMI (HVM)', is highlighted.

Launch instance wizard | EC2 Manager

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

AWS Services Resource Groups

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Feedback English (US)

Type here to search

© 2008 - 2020, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use

4:04 PM

Below are network related settings

Launch instance wizard | EC2 Manager

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

AWS Services Resource Groups

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances: 1 Launch into Auto Scaling Group

Purchasing option: Request Spot Instances

Network: vpc-dda874b6 (default) Create new VPC

Subnet: No preference (default subnet in any Availability Zone) Create new subnet

Auto-assign Public IP: Use subnet setting (Enable)

Placement group: Add instance to placement group

Capacity Reservation: Open Create new Capacity Reservation

IAM role: None Create new IAM role

Shutdown behavior: Stop

Cancel Previous Review and Launch Next: Add Storage

Feedback English (US)

Type here to search

© 2008 - 2020, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use

4:04 PM

Launch instance wizard | EC2 Manager

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

AWS Services Resource Groups

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encryption
Root	/dev/xvda	snap-022074bd3078559c6	8	General Purpose SSD (gp2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

Add New Volume

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. Learn more about free usage tier eligibility and usage restrictions.

Cancel Previous Review and Launch Next: Add Tags

Feedback English (US)

Type here to search

Launch instance wizard | EC2 Manager

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

AWS Services Resource Groups

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver.

A copy of a tag can be applied to volumes, instances or both.

Tags will be applied to all instances and volumes. Learn more about tagging your Amazon EC2 resources.

Key	(128 characters maximum)	Value	(256 characters maximum)	Instances	Volumes
This resource currently has no tags					

Choose the Add tag button or click to add a Name tag.
Make sure your IAM policy includes permissions to create tags.

Add Tag (Up to 50 tags maximum)

Cancel Previous Review and Launch Next: Configure Security Group

Feedback English (US)

Type here to search

To identify server name

us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#LaunchInstanceWizard:

AWS Services Resource Groups

RahulShetty Ohio Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. [Learn more about tagging your Amazon EC2 resources.](#)

Key	(128 characters maximum)	Value	(256 characters maximum)	Instances	Volumes
Name	Jenkins CI Server			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Add another tag (Up to 50 tags maximum)

Cancel **Previous** **Review and Launch** **Next: Configure Security Group**

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more about Amazon EC2 security groups.](#)

Assign a security group:

- Create a **new** security group
- Select an **existing** security group

Security group name: launch-wizard-2

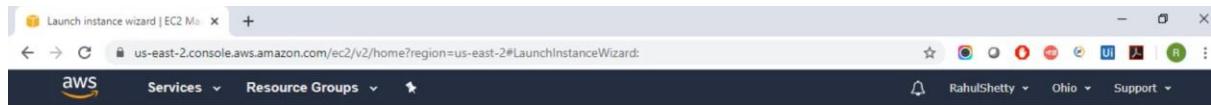
Description: launch-wizard-2 created 2020-05-23T16:05:53.780-04:00

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP f	TCP	8080	Custom 0.0.0.0/0, /0	e.g. SSH for Admin Desktop

Add Rule

Warning
Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel **Previous** **Review and Launch**



Launch Status

Your instances are now launching
The following instance launches have been initiated: i-05b7a353cf907140f [View launch log](#)

Get notified of estimated charges
Create billing alerts to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if you exceed the free usage tier).

How to connect to your instances
Your instances are launching, and it may take a few minutes until they are in the **running** state, when they will be ready for you to use. Usage hours on your new instances will start immediately and continue to accrue until you stop or terminate your instances.
Click [View Instances](#) to monitor your instances' status. Once your instances are in the **running** state, you can **connect** to them from the Instances screen. [Find out how to connect to your instances.](#)

Here are some helpful resources to get you started

- How to connect to your Linux instance
- Learn about AWS Free Usage Tier
- Amazon EC2: User Guide
- Amazon EC2: Discussion Forum

While your instances are launching you can also

A screenshot of the AWS EC2 Management Console. The top navigation bar shows "Feedback English (US)", the URL "us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances", and user details "RahulShetty", "Ohio", and "Support". The main menu on the left includes "New EC2 Experience", "EC2 Dashboard", "Events", "Tags", "Reports", "Limits", "INSTANCES" (selected), "Instances", "Instance Types", "Launch Templates", "Spot Requests", "Savings Plans", "Reserved Instances", "Dedicated Hosts", "Capacity Reservations", and "IMAGES". The central pane shows a table of instances. A search bar at the top of the table lists "search : i-05b7a353cf907140f" and "Add filter". The table columns are Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, Alarm Status, and Public DNS (IPv4). One row is selected, showing "Jenkins_CI..." as the Name, "i-05b7a353cf907140f" as the Instance ID, "t2.micro" as the Instance Type, "us-east-2b" as the Availability Zone, "running" as the Instance State, "None" as the Alarm Status, and "ec2-18-188-204-195.us-east-2.compute.amazonaws.com" as the Public DNS (IPv4). At the bottom, a detailed view of the selected instance is shown in a modal window. The modal title is "Instance: i-05b7a353cf907140f (Jenkins_CI_Server)". It has tabs for "Description", "Status Checks", "Monitoring", and "Tags". The "Description" tab displays the following details:

Instance ID	i-05b7a353cf907140f	Public DNS (IPv4)	ec2-18-188-204-195.us-east-2.compute.amazonaws.com
Instance state	running	IPv4 Public IP	18.188.204.195
Instance type	t2.micro	IPv6 IPs	-
Finding	Opt-in to AWS Compute Optimizer for recommendations. Learn more	Elastic IPs	-
Private DNS	ip-172-31-21-20.us-east-2.compute.internal	Availability zone	us-east-2b

Sessions | EC2 Management Con MobaXterm Xserver with SSH, telnet, RDP, VNC, SFTP, File, Shell, Browser, Mosh, Aws S3, WSL

mobaxterm.mobatek.net/download-home-edition.html

MobaXterm Home Edition

Download MobaXterm Home Edition (current version):

[MobaXterm Home Edition v20.2 \(Portable edition\)](#)

[MobaXterm Home Edition v20.2 \(Installer edition\)](#)

Download previous stable version: [MobaXterm Portable v20.1](#) [MobaXterm Installer v20.1](#)

You can also get early access to the latest features and improvements by downloading MobaXterm Preview version:

[MobaXterm Preview Version](#)

By downloading MobaXterm software, you accept [MobaXterm terms and conditions](#)

You can download MobaXterm and plugins sources [here](#)

If you use MobaXterm inside your company, you should consider subscribing to [MobaXterm Professional Edition](#): your subscription will give you access to professional support and to the "Customizer" software. This customizer will allow you to generate personalized your own logo, your default settings and your welcome message.

rahulshettyacademy.com

18.218.20.249 (ec2-user)

Type here to search

Terminal Sessions View Xserver Tools Games Settings Macros Help

Session Servers Tools

Quick connect...

Sessions

Name: home/ec2-user/

SSH Telnet Rsh Xdmcp RDP VNC FTP SFTP Serial File Shell Browser Mosh Aws S3 WSL

Secure Shell (SSH) session

Choose a session type...

OK Cancel

Remote monitoring

Follow terminal fold

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

rahulshettyacademy.com

Windows Type here to search

4:13 PM

rahulshettyacademy.com

4:15 PM

Instances | EC2 Management Console | MobaXterm Xserver with SSH, telnet, RDP, VNC, SFTP, RSH, Xdmcp, RDC, File, Shell, Browser, Mosh, Aws S3, WSL | us-east-2.console.aws.amazon.com/ec2/v2/home?region=us-east-2#Instances;search=i-05b7a353cf907140f;sort=descinstanceid | +

Services Resource Groups ▾

New EC2 Experience Tell us what you think

Launch Instance Connect Actions ▾

EC2 Dashboard New

Events New

Tags

Reports

Limits

Instances Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts New

Capacity Reservations

Images

Feedback English (US)

MobaXterm_Install....zip

18.218.20.249 (ec2-user)

Type here to search

Terminal Sessions View Xserver Tools Games Settings Macros Help

Session Servers Tools

Quick connect...

Sessions

Name .ssh .bash_logout .bash_profile .bashrc

Tools

Macros

SSH Telnet Rsh Xdmcp RDP VNC FTP SFTP Serial File Shell Browser Mosh Aws S3 WSL

Session settings

Basic SSH settings

Remote host 18.188.204.195 Specify username ec2-user Port 22

Advanced SSH settings

X11-Forwarding Compression Remote environment: interactive shell Execute command: Do not exit after command ends SSH-browser type: SFTP protocol Follow SSH path (experimental) Use private key C:\Users\Owner\Downloads\rahulp Adapt locales on remote server Execute macro at session start: <none>

OK Cancel

UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>

rahulshettyacademy.com

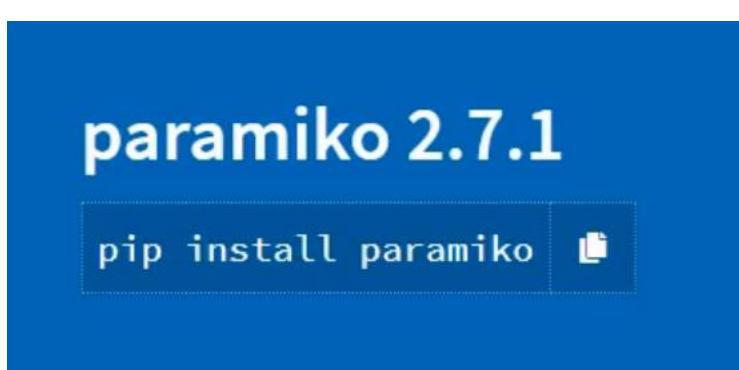
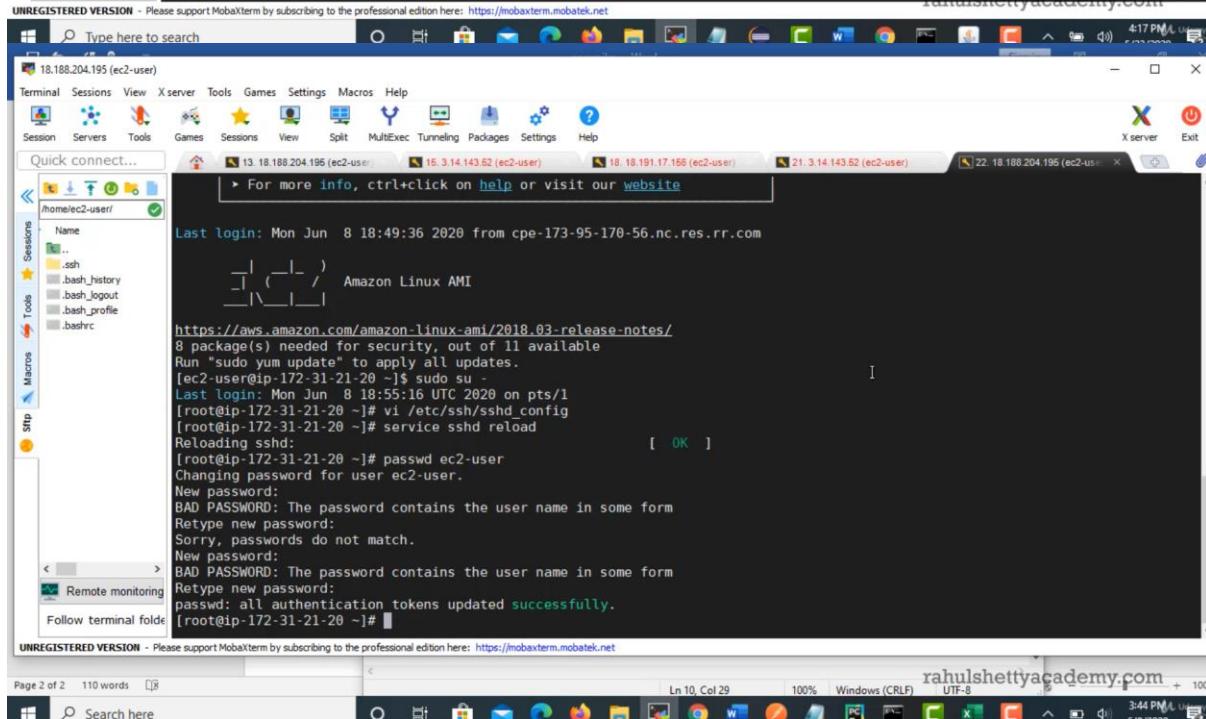
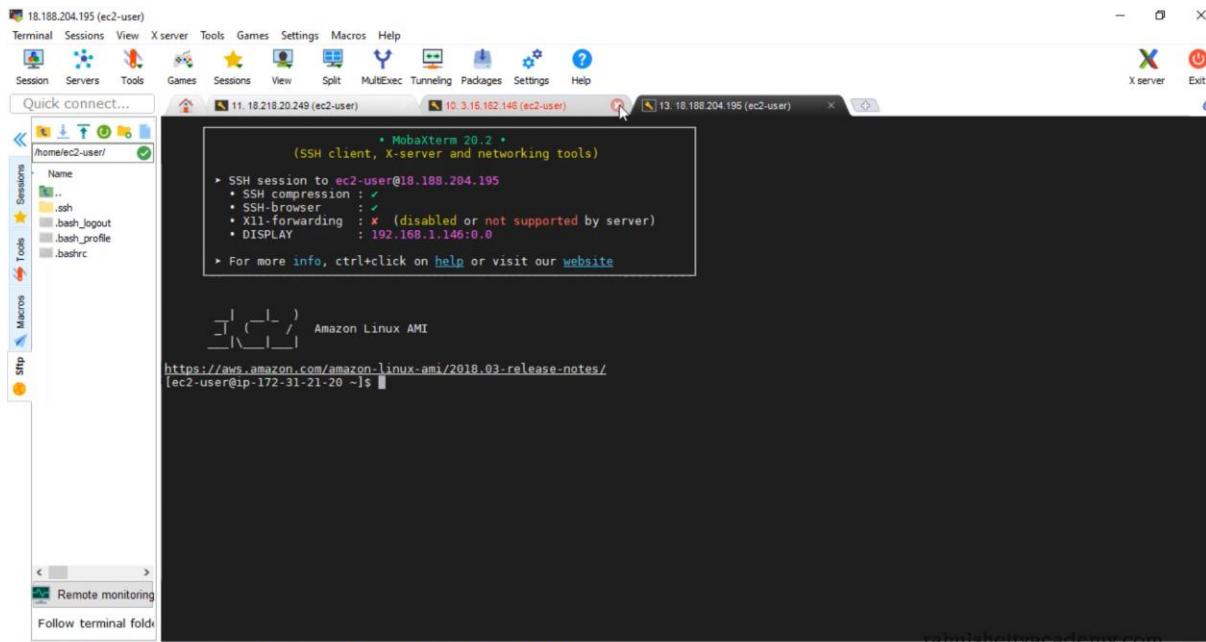
rahulshettyacademy.com

4:15 PM 4:17 PM

Windows Type here to search

rahulshettyacademy.com

4:17 PM



Q6. Execute Commands on Linux Servers from your local machine using Python Code

The screenshot shows the PyCharm IDE interface with two code editors and a terminal window.

Code Editor 1:

```
import paramiko as paramiko
from utilities.configurations import *
# Start Connection
username = getConfig()['Server']['username']
password = getConfig()['Server']['password']
host = getConfig()['Server']['host']
port = getConfig()['Server']['port']
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect(host, port, username, password)

# Run commands
```

Code Editor 2:

```
# Run commands
stdin, stdout, stderr = ssh.exec_command("ls -a")
print(stdout.readlines())
```

Terminal Output:

```
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/PycharmProjects1/BackE
['.\n', '..\n', '.bash_history\n', '.bash_logout\n', '.bash_profile\n', '.bashrc\n', '.ssh\n']
```

Terminal Session:

```
[ec2-user@ip-172-31-21-20 ~]$ ls
[ec2-user@ip-172-31-21-20 ~]$ ls -a
. .. .bash_history .bash_logout .bash_profile .bashrc .ssh
[ec2-user@ip-172-31-21-20 ~]$ vi demofile
[ec2-user@ip-172-31-21-20 ~]$ cat demofile
hello how are you?
[ec2-user@ip-172-31-21-20 ~]$
```

Code Editor 3:

```
#stdin,stdout,stderr = ssh.exec_command("ls -a")
stdin,stdout,stderr = ssh.exec_command("cat demofile")
print(stdout.readlines())
```

Terminal Output:

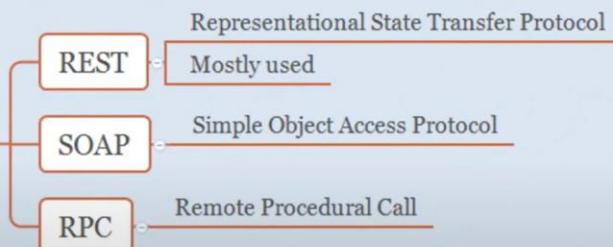
```
C:\Users\Owner\PycharmProjects1\BackEndAutomation\venv\Scripts\python.exe C:/Users/Owner/Py
['hello how are you?\n', '7\n', '120\n']
```

Message:

```
Process finished with exit code 0
```


API Protocol Types (API Testing - Part 8)

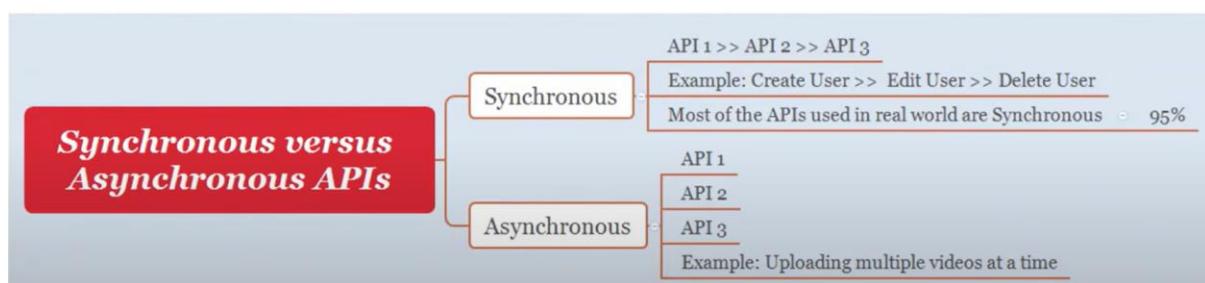
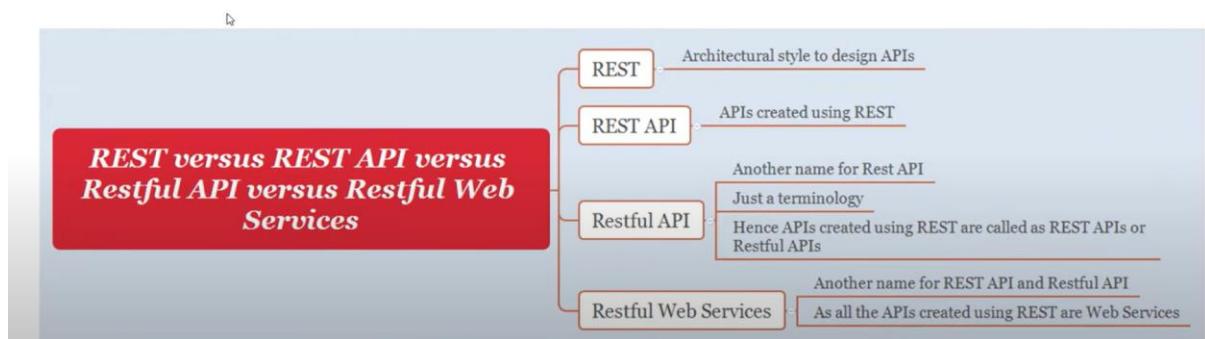
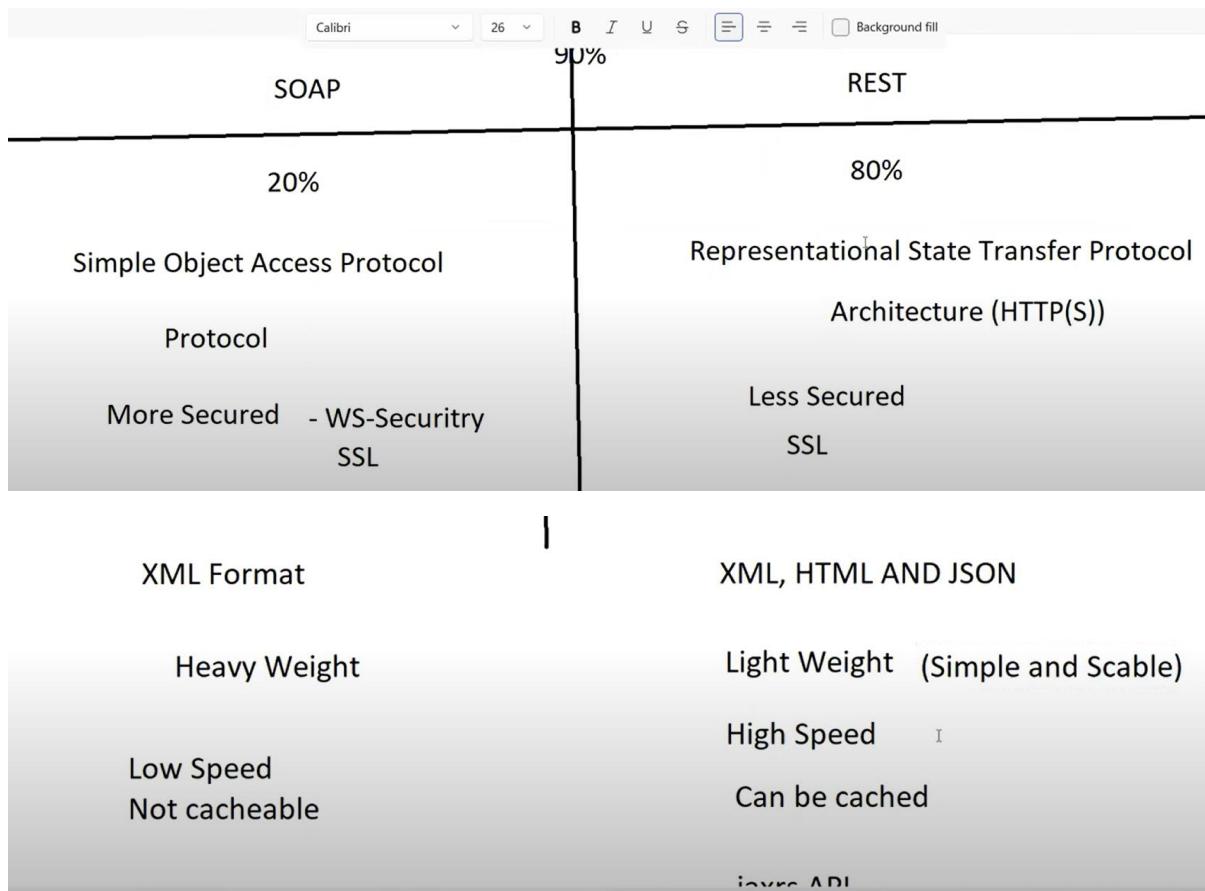
API Protocol Types



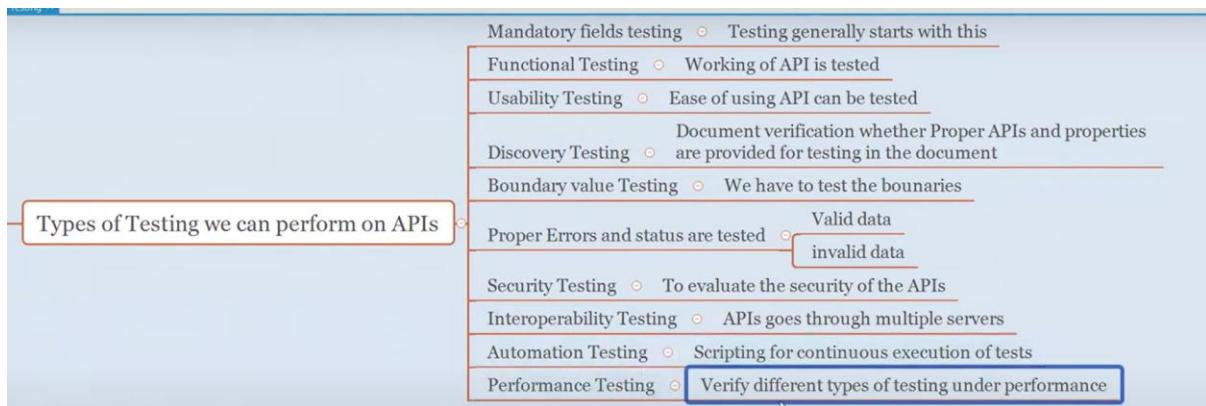
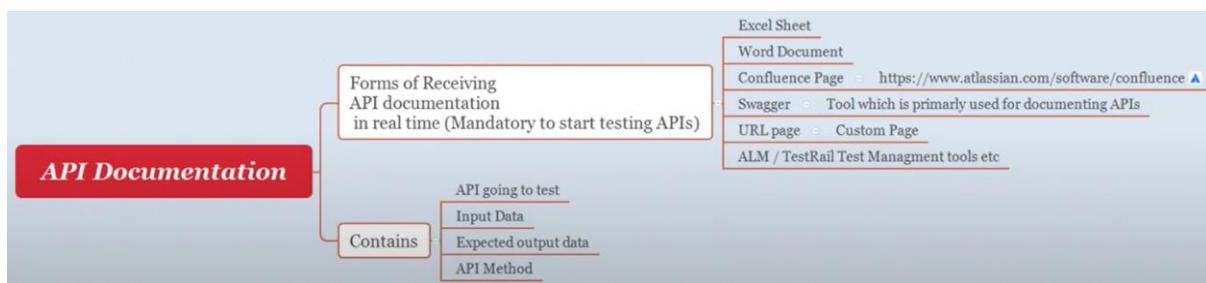
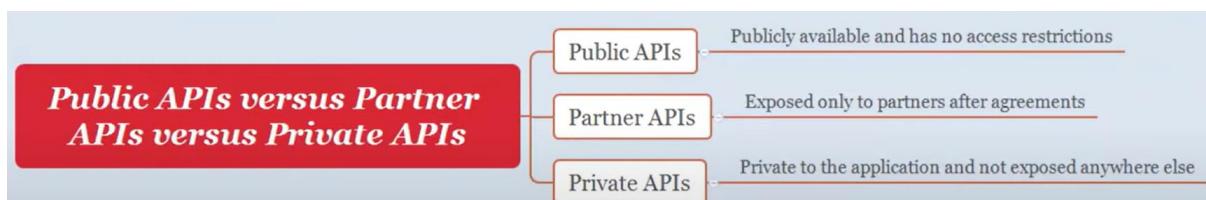
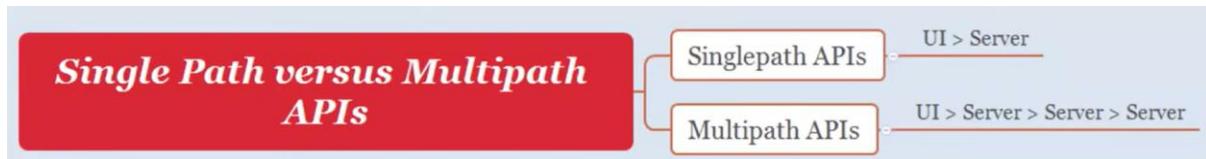
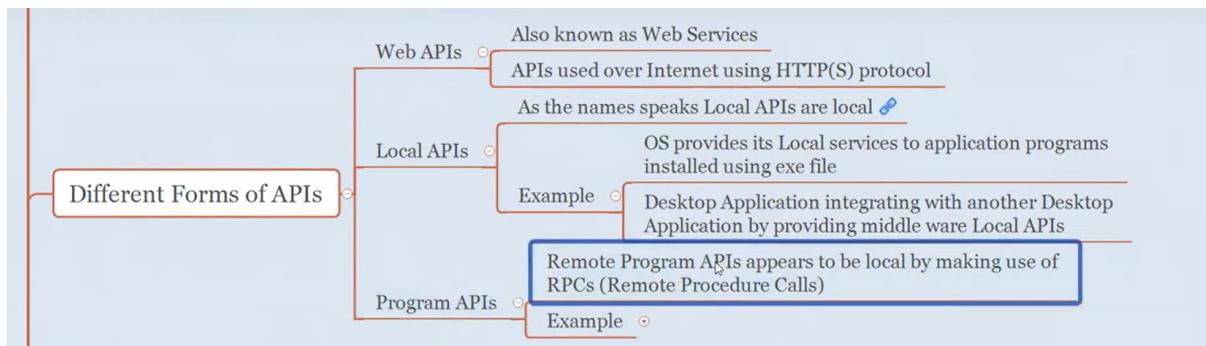
90% of Projects use SOAP and REST

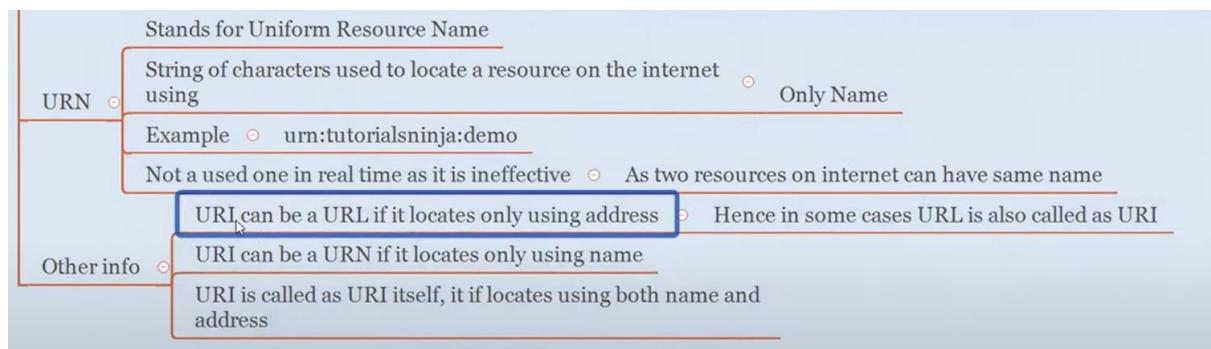
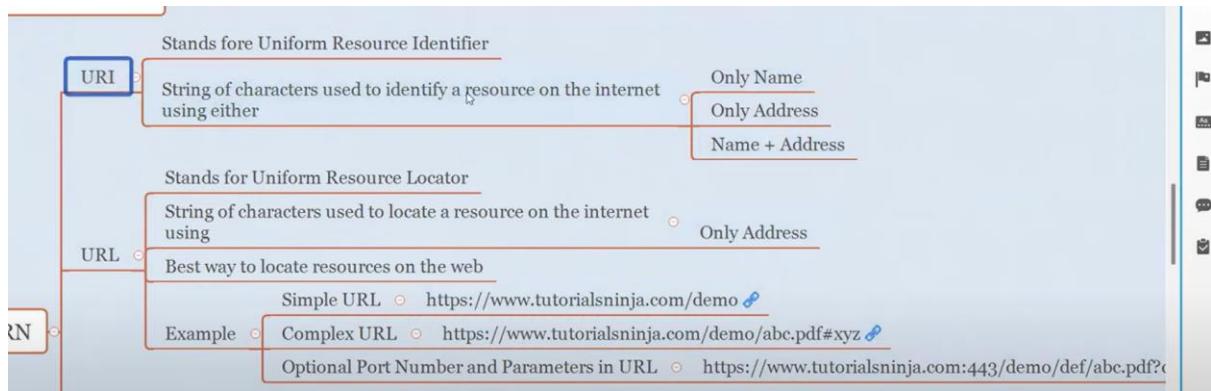
Simple Object Access Protocol	
Protocol	Has its own protocol unlike REST
More Secured than REST	Supports WS-Security and SSL
Only XML Data format	
Heavy weight as it requires more resources and bandwidth	
Low Speed	Can not be cached
Government, Banking and Payment projects etc. use SOAP	20% SOAP when compared to REST
jaxws	Java API for SOAP

Representational State Transfer	
Architecture	Uses HTTP Protocol
Less Secured than SOAP	Supports SSL
XML, HTML, JSON Data formats	
REST	Light weight as it requires few resources and less bandwidth
	High Speed
	Can be cached
Other Projects use REST	80% compared to SOAP
Known for Simplicity	Simple to build and scale
jaxrs	Java API for Rest



API3 is dependent on api2 and it depends on1 synchronous





<https://www.youtube.com/watch?v=ahhGS3fyU00&list=PLsjUcU8CQXGFqvw5OIKEZ9UUehLOtZwZO&index=21>