Section 1: Introduction to API Testing and Postman

Gain a foundational understanding of API testing’s importance and role in software development. Learn about Postman, a powerful API testing tool, and how to set it up. Explore its user interface, and create workspaces, collections, and requests for effective API testing.

Topic: Understanding the role of API testing in software development

Question 1: Why is API testing crucial in software development, and how does it differ from other testing types?

Answer: API testing ensures the functionality and reliability of the application’s APIs. Unlike UI testing, it focuses on the interface between components, making it effective for catching integration issues early. API testing also aids in performance, security, and functionality validation before the UI is developed, accelerating the development cycle.

Question 2: Give an example of a real-world scenario where API testing played a pivotal role in preventing a major software issue.

Answer: In a banking application, API testing helped detect a vulnerability where improper authorization allowed unauthorized users to access sensitive financial data. By identifying this issue early, the development team was able to fix the security flaw before it led to a data breach.

Question 3: How does API testing contribute to maintaining backward compatibility in software systems?

Answer: API testing ensures that changes to an API do not break the existing functionality for applications that rely on it. By running API tests against previous versions and new releases, developers can identify compatibility issues and adjust the API design or implementation to maintain backward compatibility.

Question 4: What are the key challenges in API testing, and how can they be mitigated?

Answer: Challenges include handling authentication, managing versioning, and dealing with complex data formats. These can be addressed by using tools like Postman to manage authentication tokens, implementing versioning strategies, and leveraging data-driven testing techniques to handle diverse data scenarios effectively.

Question 5: How does API testing impact the overall quality of a software product?

Answer: API testing ensures that components work seamlessly together, reducing integration risks. By validating data exchange, error handling, and performance, API testing enhances the software’s reliability, stability, and security, contributing to an improved user experience.

Topic: Introduction to Postman as a versatile API testing tool

Question 1: What are the primary features that make Postman a valuable tool for API testing?

Answer: Postman offers an intuitive interface for creating, sending, and validating API requests. Its features include request organization, environment management, automated testing, scripting capabilities, and collaboration tools, making it a comprehensive solution for API testing.

Question 2: How does Postman facilitate the process of parameterizing API requests for data-driven testing?

Answer: Postman allows the use of variables and environment files to dynamically substitute values in requests. By defining variables for different scenarios in an environment file, testers can execute the same request with various data inputs, enhancing test coverage and efficiency.

Question 3: Explain how Postman’s Collection Runner feature contributes to efficient API testing.

Answer: The Collection Runner enables the execution of multiple API requests in a sequence, streamlining the testing process. Testers can automate the execution of test scripts against various endpoints and input data, enabling thorough testing of API workflows.

Question 4: In what situations would you use Postman’s scripting capabilities, and how does it enhance API testing?

Answer: Postman’s scripting feature allows you to write JavaScript code for custom validation, data extraction, and manipulation. It’s useful for handling dynamic responses, extracting tokens, and performing complex calculations, thereby enabling more advanced and precise API testing scenarios.

Question 5: How does Postman foster collaboration among team members during API testing and development?

Answer: Postman provides Workspaces where team members can share collections, environments, and test scripts. Collaboration features like comments and version history allow testers, developers, and other stakeholders to communicate, review, and improve API tests collectively, ensuring higher-quality software.

Section 2: API Testing Fundamentals

Dive into the basics of APIs and HTTP/HTTPS protocols. Discover various API request types, handling headers, query parameters, and authentication methods. Develop skills to effectively handle authentication within Postman for secure API testing.

Topic: Fundamentals of APIs and HTTP/HTTPS protocols

Question 1: Imagine you are testing an e-commerce application’s API. Describe the key steps you would take to ensure proper authentication and secure data transmission over HTTPS.

Answer: To ensure secure authentication and data transmission, I would:

Verify that the API uses HTTPS to encrypt data during transmission.

Check for proper authentication methods such as OAuth, API keys, or tokens.

Confirm that sensitive data like passwords is not exposed in URLs.

Use tools like Postman to intercept and inspect requests and responses for security vulnerabilities.

Question 2: In an API testing scenario, explain the significance of request and response headers. Provide an example where headers play a critical role.

Answer: Request and response headers convey metadata and instructions between the client and server. For instance, the “Content-Type” header specifies the format of the request or response data. In a scenario involving image uploading, setting the “Content-Type” header to “multipart/form-data” indicates that the request contains file data, ensuring proper handling by the server.

Question 3: You’re testing an API that returns paginated data. How would you verify the pagination mechanism and ensure that all data is correctly retrieved?

Answer: I would send a series of requests to the API, incrementing the page parameter and monitoring the response. For each page, I’d validate the expected data, ensure that the “Next Page” link or header is accurate, and check for the presence of a “Total Pages” header or field. This approach ensures that the pagination mechanism functions correctly and that no data is missed.

Question 4: Describe a situation where an API request might require a query parameter, and explain how you would construct and test the request.

Answer: Let’s consider a weather forecast API. To retrieve the weather for a specific location and date, I would construct a GET request with query parameters like “city” and “date.” For instance, GET /forecast?city=NewYork&date=2023-08-15. I would test this request by sending it through Postman, validating the response for the correct weather data for the specified location and date.

Question 5: How does API testing contribute to ensuring data integrity and consistency in a distributed system?

Answer: API testing verifies that data is accurately exchanged between components, maintaining consistency and integrity. By sending requests with various data inputs and validating responses, API testing helps identify data discrepancies, ensuring that data remains accurate and synchronized across different parts of the system.

Topic: Different types of API requests: GET, POST, PUT, DELETE, etc.

Question 1: Explain the main differences between a GET request and a POST request in API testing. Provide a real-world example for each.

Answer: GET requests retrieve data from the server, typically using query parameters. For example, fetching a list of products (GET /products) from an e-commerce API. POST requests send data to the server, often used for creating resources, like adding a new product (POST /products) to the catalog.

Question 2: You’re testing an API that updates user profile information. How would you construct and test a PUT request to update a user’s email address?

Answer: I would construct a PUT request to the user’s profile endpoint (PUT /users/{userId}) with the updated email address in the request body. For example, { “email”: “new.email@example.com” }. I would then test the request using Postman, ensuring that the response confirms the successful update and that the user’s email has been changed accordingly.

Question 3: Describe a scenario where a DELETE request might be used in API testing, and explain the steps you would take to verify its functionality.

Answer: Consider a social media platform. To delete a user’s post, I would send a DELETE request to the post’s endpoint (DELETE /posts/{postId}). I would verify its functionality by first confirming the existence of the post, then sending the DELETE request, and finally checking that the post is no longer accessible and returns an appropriate response code (e.g., 204 No Content).

Question 4: How does API testing assist in verifying the idempotency of PUT and DELETE requests? Provide an example.

Answer: API testing ensures that repeating the same PUT or DELETE request produces the same result and doesn’t lead to unintended changes. For instance, sending the same DELETE request twice should result in the same outcome—deleting the resource. Testing this involves sending the DELETE request twice and verifying that the second request doesn’t cause any unexpected side effects or errors.

Question 5: Explain a situation where a PATCH request might be preferred over a PUT request, and outline the steps you would take to test a PATCH request.

Answer: PATCH requests are suitable for partial updates to a resource, while PUT requests replace the entire resource. Suppose you’re updating a user’s profile where only the profile picture changes. A PATCH request (PATCH /users/{userId}) would be more appropriate. To test it, I’d send a PATCH request with the changed data and validate that the specified fields are updated while others remain unchanged, using assertions in Postman.

Section 3: Writing Effective API Tests in Postman

Master the art of crafting well-structured API requests using Postman. Learn to validate responses using assertions, conduct data-driven testing with variables, and automate tests using Collection Runner and Newman CLI. Harness the power of scripts and variables for dynamic data extraction and request chaining.

Topic: Creating and structuring API requests within Postman

Question 1: You need to test an API that requires multiple steps to perform a specific action. How would you structure your Postman collection to ensure clarity and efficiency in testing this workflow?

Answer: I would structure the collection with multiple requests, each representing a step in the workflow. For example, a multi-step checkout process in an e-commerce app might involve adding items to the cart, selecting shipping, and making a payment. Each step would be a separate request within the collection, with variables to pass data between requests and ensure a coherent and organized test flow.

Question 2: Explain the benefits of using Postman environments for managing variables in your API tests. Provide an example scenario.

Answer: Postman environments allow you to store and manage variables for different test scenarios. For instance, you could have environments for testing on development, staging, and production servers. When testing an API across these environments, you can switch environments in Postman to use different base URLs, authentication tokens, or other variables without modifying the test scripts themselves, thus ensuring flexibility and reusability.

Question 3: How would you approach testing an API that requires complex authentication, such as OAuth2, within Postman? Walk through the process step by step.

Answer: To test an OAuth2-secured API in Postman:

Set up an environment with OAuth2-related variables (client ID, client secret, token URL).

Create a request for obtaining an access token using OAuth2, setting the necessary authorization type and headers.

Use the access token in subsequent requests by referencing the environment variable in the request headers.

Configure token refresh if required, ensuring seamless testing without manual token regeneration.

Validate responses for expected behavior.

Question 4: Describe a scenario where you need to handle a dynamic response from an API in your tests. How would you extract and use dynamic data using Postman’s scripting capabilities?

Answer: Imagine testing an API that returns a unique order ID upon successful order creation. I would use Postman’s scripting feature to extract this dynamic order ID from the response using JavaScript, store it in a variable, and then use that variable in subsequent requests to fetch or modify the order. This ensures accurate testing of specific order details without manual intervention.

Question 5: How can you ensure that your API tests in Postman are organized and easily maintainable, especially as the number of requests increases?

Answer: To maintain organization and manageability:

Group related requests into folders within the collection.

Use meaningful request and folder names.

Utilize comments within requests to provide context and details.

Leverage environments to store variables and configurations.

Regularly update and review the collection as the API evolves, removing redundant or outdated requests.

Topic: Automating tests using Postman’s Collection Runner and Newman CLI tool

Question 1: What is the Collection Runner in Postman, and how does it enhance the efficiency of your API testing workflow?

Answer: The Collection Runner allows you to execute a sequence of requests from a collection. It’s especially useful for data-driven testing, enabling you to iterate over sets of variables and input data. For instance, when testing an API’s CRUD operations, the Collection Runner can execute multiple scenarios with different data, automating extensive testing without manual repetition.

Question 2: Describe a situation where you need to perform load testing using Postman’s Collection Runner. How would you approach this, and what key metrics would you monitor?

Answer: Let’s say you’re testing an e-commerce API during a flash sale event. Using the Collection Runner, you would send concurrent requests to simulate high user traffic. Monitor key metrics like response times, throughput, and error rates. Ensure that the API can handle the load without performance degradation or crashes, allowing shoppers to have a smooth experience.

Question 3: Explain the purpose of the Newman CLI tool in Postman and how it can be integrated into a continuous integration (CI) process.

Answer: Newman is a command-line tool that allows you to run Postman collections outside of the Postman interface. It’s perfect for integrating API tests into CI pipelines. By invoking Newman in your CI configuration, you can ensure that your API tests are automatically executed whenever changes are made to the codebase, helping maintain code quality and preventing regressions.

Question 4: How can you parameterize data in your API tests when using the Collection Runner or Newman? Provide an example.

Answer: Parameterizing data involves using variables that change for each iteration of the test. For instance, in an e-commerce scenario, you might have variables like product IDs, quantities, and user details. By creating a CSV or JSON file with different data sets and using Newman’s -d flag, you can execute the collection with various data inputs, ensuring comprehensive test coverage.

Question 5: In a scenario where you have complex test dependencies, how can you ensure the proper order of execution when running tests using the Collection Runner or Newman?

Answer: You can use Postman’s test scripts to manage dependencies and ensure the correct order of execution. In each request’s test script, set environment variables that act as flags. Use these flags in subsequent requests’ scripts to conditionally run requests based on the completion of previous steps. This way, you orchestrate the desired order of execution and ensure accurate test scenarios.

Section 4: Advanced API Testing Techniques

Explore advanced techniques such as mocking APIs with Postman Mock Servers, testing pagination, rate limiting, and error scenarios. Gain insights into load testing and performance testing using Postman, and learn how to handle complex scenarios like session-based APIs.

Topic: Mocking APIs using Postman Mock Servers

Question 1: Describe a situation where you would use Postman Mock Servers for API testing. How would you set up and validate the mock server’s behavior?

Answer: Consider testing a mobile app’s registration flow. To simulate the backend’s response during development, I’d use a Postman Mock Server. I’d create a mock request and response, define response codes, headers, and example data. Then, I’d configure the app to make requests to the mock server instead of the actual API. By monitoring the mock server’s responses, I can ensure the app handles different scenarios correctly.

Question 2: How can you simulate dynamic behavior, such as time-based or conditional responses, using Postman Mock Servers?

Answer: To simulate time-based responses, I’d use Postman’s scripting feature to calculate and adjust timestamps in the response based on the current time. For conditional responses, I’d set up logic in the script to analyze request parameters and return different responses accordingly. This approach ensures realistic testing of time-sensitive or context-dependent scenarios.

Question 3: Explain the process of transitioning from using a Postman Mock Server for testing to the actual API for production. What steps are involved?

Answer: Transitioning involves updating the app’s configuration to point to the actual API endpoint instead of the mock server. Steps include:

Replacing the API base URL in the app code.

Testing thoroughly to verify that the app behaves correctly with the real API.

Gradually phasing out the mock server in favor of the actual API, while monitoring and addressing any issues that arise.

Question 4: How can you use Postman’s Mock Server to simulate different response codes (e.g., 404, 500) and verify how your application handles errors?

Answer: In Postman Mock Server, I’d configure responses for different error codes (e.g., 404 or 500) and include error messages or details in the response body. By directing the app to the mock server and making requests that trigger these error scenarios, I can ensure the app handles errors gracefully, displays appropriate messages, and doesn’t break.

Question 5: What are the benefits of using Postman Mock Servers over directly testing with the actual API during development?

Answer: Postman Mock Servers offer advantages such as:

Early and parallel development by frontend and backend teams.

Isolation for frontend development from backend dependencies.

Rapid testing of various scenarios without impacting the actual API.

Realistic simulation of different responses for thorough testing.

Reduced reliance on the availability and stability of the actual API during development.

Topic: Testing pagination, rate limiting, and error scenarios

Question 1: How would you design a comprehensive test suite to validate the pagination functionality of an API? What factors would you consider?

Answer: I would design tests to cover scenarios like:

Initial request and response validation.

Verifying that the “Next Page” link or token is correctly provided.

Testing pagination through a large dataset to ensure proper results.

Evaluating response time and performance as the dataset size increases.

Question 2: Describe a strategy for testing rate limiting in an API using Postman. How would you simulate exceeding rate limits and verify the API’s behavior?

Answer: I’d create a test suite that sends a high volume of requests in a short time, exceeding the rate limit. To simulate different responses, I’d use Postman’s scripting to alternate between 429 (Too Many Requests) and successful responses based on the request count. By analyzing the API’s responses, I can confirm that it correctly enforces rate limits and handles exceeded limits.

Question 3: Explain how you would approach testing different error scenarios (e.g., 400 Bad Request, 401 Unauthorized) in an API. Provide an example test case and the expected outcomes.

Answer: For a 401 Unauthorized scenario:

Test case: Send a request without valid authentication.

Expected outcome: Verify the response code is 401, and the response body includes an appropriate error message, such as “Authentication required.”

Question 4: What challenges might arise when testing error scenarios, and how can you overcome them?

Answer: Challenges include:

Ensuring consistency in error responses across different endpoints.

Handling edge cases and corner scenarios that trigger specific errors.

Verifying that error responses include detailed and helpful messages.

To overcome these challenges, I’d create a centralized library of expected error responses, automate tests for common error scenarios, and perform manual tests for unique situations.

Question 5: How can you use Postman’s scripting capabilities to automate testing of error scenarios? Provide an example.

Answer: To test an error scenario like a 404 Not Found:

Use a script in the pre-request section to modify the request URL to a non-existent endpoint.

Send the request and verify that the response code is 404.

In the tests section, validate that the response body contains an appropriate error message, indicating the resource was not found.

By effectively testing pagination, rate limiting, and error scenarios, you ensure your API handles various situations gracefully and provides a robust user experience.

Section 5: Integration and Collaboration

Discover ways to collaborate effectively by using Postman Workspaces for version control and team collaboration. Learn about importing, exporting, and sharing collections, and integrate Postman into CI/CD pipelines. Finally, generate API documentation and follow best practices for maintaining organized and efficient API test suites.

Topic: Version control and collaboration using Postman Workspaces

Question 1: How does using Postman Workspaces contribute to effective version control and collaboration in API testing? Provide an example of a scenario where Workspaces prove beneficial.

Answer: Postman Workspaces provide a shared environment where team members can collaborate on API testing. When multiple testers work on different features, Workspaces keep their collections organized and separate. For example, in a fintech app, one team member might focus on payment APIs, while another works on account management. Workspaces ensure that changes made by one person don’t impact another’s work, maintaining version control and efficient collaboration.

Question 2: Describe how Postman Workspaces handle conflict resolution when multiple team members are editing the same collection simultaneously.

Answer: Postman detects conflicts when team members make conflicting changes to the same collection. When a conflict occurs, Postman presents a resolution screen where users can compare changes, choose which version to keep, and merge modifications. This ensures that collaborative edits are managed without overwriting or losing work, promoting seamless teamwork.

Question 3: In a situation where a team member introduces an unintentional issue in a collection, how can you use Postman Workspaces to mitigate the problem and maintain the overall quality of API testing?

Answer: If an issue is introduced, you can revert to a previous version of the collection from the version history in Postman Workspaces. This allows you to roll back to a working state and continue testing. Additionally, you can involve team members in discussions within Postman comments, addressing the issue and collaborating to find a solution.

Question 4: How can you ensure that different team members have the appropriate access levels and permissions within a Postman Workspace?

Answer: Postman Workspaces provide granular access control. To ensure appropriate permissions, assign roles like “Read,” “Write,” or “Admin” to each team member. For instance, a tester might have write access to create and modify tests, while a developer might have admin access to manage collections. This prevents unauthorized changes and maintains a controlled testing environment.

Question 5: Explain how Postman Workspaces integrate with version control systems like Git. How does this integration facilitate seamless collaboration and code review?

Answer: Postman Workspaces can be linked to a Git repository, allowing collection changes to be automatically synchronized with the repository. When a team member makes changes in Postman, they can create a pull request to merge those changes into the Git repository. This integration streamlines collaboration, enabling code review, and ensuring that API tests are aligned with the codebase.

Topic: Importing and exporting collections for sharing and backup

Question 1: How can you efficiently share a Postman collection with a colleague who is not part of your Postman Workspace? Describe the steps and available options.

Answer: To share a collection externally, you can:

Export the collection as a JSON file.

Share the JSON file with your colleague through email or a shared drive.

Your colleague can then import the JSON file into their Postman instance, accessing the collection and its requests.

Question 2: Explain a scenario where exporting and importing collections become crucial for ensuring continuity during a project transition or migration.

Answer: Imagine transitioning from a development to a testing environment. Exporting the collection from the development workspace and importing it into the testing workspace ensures that the same set of API tests is available, maintaining consistency and reducing the risk of missing tests during the transition.

Question 3: How would you handle the situation where you need to update a shared Postman collection after it has been exported and shared with colleagues?

Answer: To update a shared collection, you can follow these steps:

Make the necessary changes to the collection within your Postman Workspace.

Export the updated collection as a new JSON file.

Share the updated JSON file with your colleagues, informing them of the changes.

They can then import the new JSON file to receive the updated collection in their Postman instances.

Question 4: What precautions should you take when exporting collections that include sensitive data, such as authentication tokens or API keys?

Answer: To ensure security when exporting collections with sensitive data:

Before exporting, review the collection to ensure no sensitive data is included.

Use Postman’s environment feature to store sensitive variables separately from the collection.

When sharing, inform colleagues to configure their environments with their own sensitive data.

Question 5: How does importing and exporting collections aid in disaster recovery and backup strategies for your API testing efforts?

Answer: Exporting collections provides a backup of your API tests, scripts, and configurations. In case of data loss or system failures, you can import the backed-up collection to quickly restore your testing setup. This ensures continuity and minimizes downtime, allowing you to resume testing promptly.

API testing has been considered the future of software testing and it has advantages in the ability to test for Backend functionality, time effectiveness, language independence, and GUI integration. It becomes a must for software testing projects to ensure product quality and at the same time important part of Testing/SDET interview questions and answers. In the post below we will check API Testing Interview Questions and Answers along with real time examples.

Content will be focused on below :  
- REST API testing interview Questions  
- Advanced API testing Interview Questions  
- Rest Assured API Testing interview Questions for experienced

WHAT IS AN API?

Imagine we are sitting at a table in a restaurant with a menu of choices to order from. The kitchen is the part of the “system” that will prepare your order. What is missing is a critical link to communicate your order to the kitchen and deliver your food back to your table. That’s where the waiter or API comes in. The waiter is the messenger – or API – that takes your request or order and tells the kitchen – the system – what to do. Then the waiter delivers the response back to you; in this case, it is the food.

Here is a real-life API example. We may be familiar with the process of searching for flights online. Just like the restaurant, you have a variety of options to choose from, including different cities, departure and return dates, and more. Let us imagine that you’re booking you are flight on an airline website.

We choose departure city and date, return city and date, cabin class, as well as other variables. In order to book your flight, you interact with the airline’s website to access their database and see if any seats are available on those dates and what the costs might be.

However, what if we are not using the airline’s website? What if we are using an online travel service, such as Kayak or Expedia, which aggregates information from a number of airline databases?

The travel service, in this case, interacts with the airline’s API. The API is the interface that, like your helpful waiter, can be asked by that online travel service to get information from the airline’s database to book seats, baggage options, etc. The API then takes the airline’s response to your request and delivers it right back to the online travel service, which then shows you the most updated, relevant information.

VERY IMP:  [BEGINNERS GUIDE FOR API TESTING](https://www.linkedin.com/pulse/beginners-guide-api-testing-sidharth-shukla%3FtrackingId=EfiAvt%252F3RPWRAonK1DAlBg%253D%253D/?trackingId=EfiAvt%2F3RPWRAonK1DAlBg%3D%3D)

Explain the major components of an API?

<https://www.linkedin.com/posts/sidharth-shukla-77b53145_sidpost-api-software-activity-7089077112594182146-LpDw?utm_source=share&utm_medium=member_desktop>

21. How to use Basic authentication in automation?

  Response resp = given()  
   .auth()  
   .basic("sid", "sid").when().get("https://reqres.in/api/users/2");  
     
    22. How to use Pre-emptive authentication in automation?

  Response resp1 = given()  
   .auth()  
   .preemptive().basic("sid", "sid").when().get("https://reqres.in/api/users/2");  
     
    23. How to use digest authentication in automation?

  Response resp2 = given()  
   .auth()  
   .digest("sid", "sid").when().get("https://reqres.in/api/users/2");  
      
    24. How to use Oauth2 authentication in automation?

  Response resp3 = given()  
   .auth()  
   .oauth2("").when().get("https://reqres.in/api/users/2");  
      
    25. How to use Oauth authentication in automation?    
      
   Response resp4 = given()  
   .auth()  
   .oauth("consumerKey", "consumerSecret", "accessToken", "secretToken").when().get("https://reqres.in/api/users/2");  
     
    26. How to use header for authorization(oauth2) in automation?  
  
   Response resp5 = given().header("Authorization","accessToken")  
   .when().get("https://reqres.in/api/users/2");  
     
    }

  29. What are the main differences between API and Web Service?

- All Web services are APIs but not all APIs are Web services.

- Web service uses three styles of use: SOAP, REST and XML-RPC for communication whereas   API  may be exposed in multiple ways.

  -  Web service needs a network to operate but APIs don’t need a network to operate.

Explain API Chaining with examples ?

<https://www.linkedin.com/posts/sidharth-shukla-77b53145_sidpost-apitesting-testautomation-activity-7118075315326746624-tJQh?utm_source=share&utm_medium=member_desktop>

 30. What is REST?  
  
- REST (Representational State Transfer) is an architectural style for developing web services which exploit the ubiquity of HTTP protocol and uses HTTP method to define actions. It revolves around resource where every component being a resource that can be accessed through a shared interface using standard HTTP methods.

- In REST architecture, REST Server provides access to resources and client accesses and makes these resources available.

- Each resource is identified by URIs or global IDs, and REST uses multiple ways to represent a resource, such as text, JSON, and XML.

What all 4xx status code you have used in your project, can you explain some of the error codes?  
  
<https://www.linkedin.com/posts/sidharth-shukla-77b53145_testing-testautomation-automation-activity-7091624769585905664-eUKK?utm_source=share&utm_medium=member_desktop>

Explain what statelessness means in REST?

Ans: Statelessness means that the client and server don’t store information about each other’s state. Since the server stores no information, it treats each client request as a new request.

As a consequence, the following conditions would apply:

The client request contains all information required for the server to process the request

Client application is responsible for storing session state

Explain different types of parameters in API, like path param, query param ?

<https://www.linkedin.com/posts/sidharth-shukla-77b53145_java-automation-javavirtualmachine-activity-7080762585028956161-qS-V?utm_source=share&utm_medium=member_desktop>

 How to send a Nested JSON object as payload?

Ans:  
Ex: {“space”: {

“name”:”myspace”,

“id”:”sidharth″

}}

HashMap<String,Object> mainobj= new HashMap<String,Object>();

HashMap<String,String> subobj= new HashMap<String,String>();

subobj.put("name","QA");

subobj.put("id","sidharth");

mainobj.put("space",subobj);

What is JSON Schema and how to perform Schema Testing with Rest Assured?  
  
Ans: [Click Here For Answer With Code](https://automationreinvented.blogspot.com/2022/03/what-is-json-schema-and-how-to-perform.html)

What are the disadvantages of REST API?

Ans:

Doesn’t enforce security practices

HTTP method limits you to synchronous requests

Due to statelessness, you might be unable to maintain state (e.g. in sessions)

 Explain Idempotency in API with examples:

<https://www.linkedin.com/posts/sidharth-shukla-77b53145_testing-sidpost-apitesting-activity-7178593745385238528-ysS4?utm_source=share&utm_medium=member_desktop>

What is Input injection and what are different ways to simulate user input?

Ans::  
Input Injection:  It is the act of simulating user input, in several ways you can simulate user input.

Direct Method Invocation

Invocation using an accessibility interface

Simulation using low-level input

Simulation using a device driver

Simulation using a robot  
  
What are some architectural styles for creating a Web API?

Ans::This is one of the fundamental Web API interview questions. Bellows are four common Web API architectural styles:

HTTP for client-server communication

XML/JSON as formatting language

Simple URI as the address for the services

Stateless communication

Which purpose does the OPTIONS method serve for the RESTful Web services?

Ans: The OPTIONS Method lists down all the operations of a web service supports. It creates read-only requests to the server.

1. What are the types of methods most used in RestAPI Testing?

GET/POST/PUT/PATCH/DELETE/HEAD/OPTIONS

P1 : GET POST DELETE

P2 : PUT

P3 : PATCH

2. What are the types of Status codes?

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

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

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

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

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

3. What are the status codes you have come across in your API testing project?

<https://automationreinvented.blogspot.com/2019/03/what-are-most-used-api-status-codes.html>

4. What is GET Method?

GET Retrieve information about the REST API resource

5. What is POST Method?

POST Create a REST API resource

6.What is PUT Method?

PUT Update a REST API resource

7. What is DELETE Method?

DELETE Delete a REST API resource or related component

8. What is HEAD method?

The HEAD method asks for a response identical to that of a GET request, but without the response body.

This is useful for retrieving meta-information written in response headers, without having to transport the entire content

9. What is OPTIONS method?

The OPTIONS method returns the HTTP methods that the server supports for the specified URL.

This can be used to check the functionality of a web server by requesting '\*' instead of a specific resource.

10. PUT VS POST

PUT vs POST : An Example

Let’s list down when to use POST and when to use PUT :

GET /user-management/users: Get all users

POST /user-management/users: Create a new user

GET /user-management/users/{id} : Get the user information identified by "id"

PUT /user-management/users/{id} : Update the user information identified by "id"

It is good practice to use:

 - PUT for UPDATE operations.

 - POST for CREATE operations.

PUT is Idempotent

POST is not Idempotent.

10. Http vs HTTPS

 In HTTP, URL begins with “http://” whereas URL starts with “https://”

HTTP uses port number 80 for communication and HTTPS uses 443

HTTP is considered to be unsecured and HTTPS is secured

HTTP Works at Application Layer and HTTPS works at Transport Layer

In HTTP, Encryption is absent and Encryption is present in HTTPS

HTTP does not require any certificates and HTTPS needs SSL Certificates

11. Automate GET method and validate the status code?

|  |
| --- |
| @Test(description="Verify status code for GET method-users/2 as 200")  public static void verifyStatusCodeGET() {    Response resp = given()                                          .when()          .get("https://reqres.in/api/users/2");   assertEquals(resp.getStatusCode(),200);    } |

12.  Automate GET method and fetch response body?

 @Test(description="Verify status code for GET method-users/2 as 200")

 public static void verifyStatusCodeGET() {

Response resp=given().when().get("https://reqres.in/api/users/2");

assertEquals(resp.getBody().asString(),200);

 }

 13. Automate GET method and verify value from response body?(validate that total number pages =12)

|  |
| --- |
| @Test(description="Verify status code for GET method-users/2 as 200")  public static void verifyStatusCodeGET() {    Response resp=given().when().get("https://reqres.in/api/users");   System.out.println(resp.path("total").toString());   assertEquals(resp.getStatusCode(),200);  assertEquals(resp.path("total").toString(),"12"); } |

 14. How to pass query param with GET method in Rest Assured?

API Query parameters can be defined as the optional key-value pairs that appear after the question mark in the URL. Basically, they are extensions of the URL that are utilized to help determine specific content or action based on the data being delivered. Query parameters are appended to the end of the URL, using a '?

|  |
| --- |
| @Test  public void validateQueryParamInGiven() {      Response resp = given().queryParam("page", "2").  when().get("https://reqres.in/api/users");   assertEquals(resp.getStatusCode(),200);   System.out.println(resp.getBody().asString());  } |

15. How to pass header for GET method in Rest Assured?

|  |
| --- |
| @Test  public void validateGivenHeader() {      Response resp = given().header("Content-Type", "application/json").  when().get("https://gorest.co.in/public-api/users");   assertEquals(resp.getStatusCode(),200);  System.out.println(resp.getBody().asString());  } |

16. How to automate PATCH method in rest Assured?

The HTTP PATCH method can be used when a resource needs to be updated. This method is especially useful if a resource is large and the changes being made are small.

|  |
| --- |
| @Test(description="validate with jsonpath and json object and pass post body as json file")  public void MethodValidationPUT() throws IOException, ParseException {      FileInputStream file = new FileInputStream(new File (System.getProperty("user.dir")+"\\TestData\\put.json"));   Response resp =  given().header("Content-Type" , "application/json").body(IOUtils.toString(file,"UTF-8")).          when().patch("https://reqres.in/api/users/2");   assertEquals(resp.getStatusCode(),200);  assertEquals(resp.path("job"),"tester");    } |

17. How to automate PUT method in Rest Assured?

A PUT method puts or places a file or resource precisely at a specific URI. In case a file or a resource already exists at that URI, the PUT method replaces that file or resource. If there is no file or resource, PUT creates a new one.

|  |
| --- |
| @Test(description="validate with jsonpath and json object and pass post body as json file")  public void MethodValidationPUT() throws IOException, ParseException {      FileInputStream file = new FileInputStream(new File (System.getProperty("user.dir")+"\\TestData\\put.json"));   Response resp =  given().header("Content-Type" , "application/json").body(IOUtils.toString(file,"UTF-8")).          when().put("https://reqres.in/api/users/2");   assertEquals(resp.getStatusCode(),200);  assertEquals(resp.path("job"),"tester");    } |

18. How to automate POST method in Rest Assured?

 POST requests are used to send data to the API server to create or update a resource. The data sent to the server is stored in the request body of the HTTP request

|  |
| --- |
| @Test(description="validate with jsonpath and json object and pass post body as json file")  public void MethodValidationPOST() throws IOException, ParseException {      FileInputStream file = new FileInputStream(new File (System.getProperty("user.dir")+"\\TestData\\put.json"));   Response resp =  given().header("Content-Type" , "application/json").body(IOUtils.toString(file,"UTF-8")).          when().post("https://reqres.in/api/users");   assertEquals(resp.getStatusCode(),201);  assertEquals(resp.path("job"),"tester");    } |

What is the major drawback of using SOAP?

Ans: When using SOAP, users often get the firewall security mechanism as the biggest obstacle. This block all the ports leaving few like HTTP port 80 and the HTTP port used by SOAP that bypasses the firewall. The technical complaint against SOAP is that it mixes the specification for message transport with the specification for message structure.

When to use contract testing?[​](https://draft.blogger.com/blog/post/edit/2695082220643154739/2959745970430611222)

Ans: Contract testing is immediately applicable anywhere where you have two services that you need to communicate - such as an API client and a web front-end. Although a single client and a single service is a common use case, contract testing really shines in an environment with many services (as is common for a microservice architecture). Having well-formed contract tests makes it easy for developers to avoid version hell. Contract testing is a killer app for microservice development and deployment.

To learn more about the use of the pact in contract testing please refer: [ContractTestingWithPact](https://draft.blogger.com/blog/post/edit/2695082220643154739/2959745970430611222)

What are the status codes you have come across in your API testing project?

Ans: [Click Here For Status Codes](https://draft.blogger.com/blog/post/edit/2695082220643154739/2959745970430611222)

How to fix the error import io.restassured.RestAssured cannot be resolved?

Do you have the scope set to test when you are adding the mentioned dependencies? This limits the code from accessing that dependency's classes within the source code. That is, we can access those classes within your testsources (ex: ${project.dir}/src/test/java/<package>, ${project.dir}/test/<package>.

If that is not your intended use case, just remove the scope attribute.

 <dependency>

      <groupId>io.rest-assured</groupId>

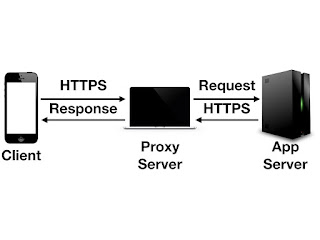
      <artifactId>rest-assured</artifactId>

      <version>4.1.1</version>

  </dependency>

74. How to check API responses in mobile (Android/IOS) browser?

Ans: Using Fiddler or Charles Proxy. A proxy server is an intermediary for requests which travel from client to server and vice-versa.  
A proxy can exist on the same machine as the client or server, it can also exist on a separate machine. This is the case for the setup we are going to use in the current context where we will have a client (mobile phone with an application we want to debug), a proxy server (our PC ) and a Server (which communicates with the client).



How to validate Query Parameters in API Testing?

<https://www.linkedin.com/posts/sidharth-shukla-77b53145_sidpost-apitesting-qualityassurance-activity-7105754145323679744-k0ai?utm_source=share&utm_medium=member_desktop>

Scenario 1:

Question:

You are tasked with testing the authentication functionality of an API using Rest Assured. How would you approach this scenario?

Answer:

Firstly, I would ensure that I have clear documentation or understanding of the authentication mechanism used by the API, whether it's basic authentication, OAuth, or token-based authentication. Then, I would write test cases using Rest Assured to verify that the authentication process works as expected. This would involve sending requests with valid credentials and ensuring that the API responds with the expected status codes and authentication tokens. Additionally, I would simulate scenarios such as providing invalid credentials or missing authentication tokens to verify that the API handles these cases appropriately, returning the correct error responses.

Scenario 2:

Question:

You need to test an API endpoint that retrieves user data based on certain criteria using Rest Assured. How would you design your test cases for this scenario?

Answer:   
To test the user data retrieval endpoint, I would first identify the criteria that can be used to filter or retrieve specific user data, such as user IDs, usernames, or other attributes. Then, I would design test cases to verify that the endpoint returns the correct user data based on different combinations of criteria. This would involve sending requests with various parameters using Rest Assured and validating that the API responds with the expected user data. I would also include test cases to verify edge cases, such as requesting data for non-existent users or providing invalid criteria, to ensure that the API handles these scenarios gracefully and returns appropriate error responses. Additionally, I would consider testing performance aspects by sending requests with different load levels to assess the endpoint's scalability and response times.

API Testing External Resource with 100+ Q&A

<https://drive.google.com/file/d/1vlhMeb0jSB9fH6Z78nMG-iDg6pP8HADn/view?usp=sharing>