50+ API Testing Interview Questions & Answers

By Rajkumar Updated on August 18, 2024

APIs are a critical part of the modern web, as they allow developers to share data across different platforms.

As a software tester, you may have been asked to test an API. If so, you'll want to know the best way to approach this task from both a technical and non-technical standpoint. In order to do so, it's important that you have a strong understanding of what APIs are and how they work.

In this blog post, we'll be going over some of the most frequently asked Web API testing interview questions and answers. We will discuss what they are and how to answer them as well as provide a list of other related topics that you should know.

Before going ahead, let's see some unavoidable Interview Questions which every hiring manager asks you in any Software Testing interview.

- What Are The Reasons For Choosing Software Testing As Your Career
- Tell Me About Yourself (Detailed Answer)

Let's move with the actual post on API Interview Questions And Answers.

Best Web API Testing Interview Questions

Before going ahead, I would like to let you know that we have another post where we covered 30+ **Postman Interview Questions** separately.

1. What is an API?

API is an acronym and it stands for **A**pplication **P**rogramming **I**nterface. API is a set of routines, protocols, and tools for building Software Applications. APIs specify how one software program should interact with other software programs.

In simple words, API stands for **A**pplication **P**rogramming **I**nterface. API acts as an interface between two software applications and allows the two software applications to communicate with each other. API is a collection of software functions that can be executed by another software program.

2. What are the types of APIs?

When discussing the types of APIs, it's essential to understand the various categories that serve different purposes in software development. Here are some of the main types of APIs:

- **Web APIs**: These are interfaces that allow interaction between web servers and clients. They commonly use HTTP protocols to facilitate communication, enabling applications to access and exchange data over the internet. Examples include RESTful APIs and GraphQL.
- **Operating System APIs**: These APIs provide functionalities for interacting with the operating system resources. They allow software applications to perform tasks such as managing files, handling memory, and managing processes. Examples include the Windows API and POSIX.
- **Library APIs**: These consist of a set of routines, protocols, and tools for building software and applications. Library APIs allow developers to use predefined functions and processes without having to reinvent the wheel, thus speeding up the development process. Examples include the C Standard Library and Java's JavaFX API.
- **Hardware APIs**: These interfaces allow software to communicate directly with hardware components. They provide the necessary commands and controls to interact with physical devices, like printers or graphics cards. Examples include the JavaScript Web API for accessing device capabilities and the Android Open Accessory Protocol.

Each type of API plays a crucial role in enabling developers to interact with different layers of technology, facilitating a smoother integration and development experience.

3. What is API Testing?

API testing is a type of software testing that involves testing APIs directly and also as a part of integration testing to check whether the API meets expectations in terms of functionality, reliability, performance, and security of an application. In API Testing our main focus will be on the Business logic layer of the software architecture. API testing can be performed on any software system which contains multiple APIs.

4. What is the difference between manual API testing and automated API testing?

Manual API testing involves testers executing test cases in an ad-hoc manner without the use of automation tools. This method is exploratory in nature and allows for a hands-on approach where testers can understand the application's functionality, examine responses in real-time, and quickly adapt test cases based on immediate findings. Although it offers flexibility, manual testing can be time-consuming and error-prone, especially when dealing with complex APIs or large datasets.

In contrast, automated API testing uses scripts and tools to run tests without human intervention. This approach is efficient for repetitive tasks, large-scale testing, and regression testing, as it ensures consistent results and reduces human error. Automated testing can significantly speed up the testing process and is particularly useful for continuous integration/continuous deployment (CI/CD) environments. However, it requires an initial investment in time and resources to develop and maintain the automated tests.

Both methods have their advantages and are often used in conjunction to maximize testing effectiveness.

5. What are the common API Testing Types?

API testing typically involves the following practices:

- **Unit testing:** This involves testing individual components or functions of the API to ensure they work as expected. It helps detect errors at an early stage in development.
- **Functional testing:** This type of testing verifies that the API behaves according to the requirements. It checks that the endpoints are returning the correct data and are functioning as intended.
- **Load testing:** Load testing evaluates the API's performance under heavy traffic or stress. It measures how well the API handles multiple requests simultaneously.
- **Runtime/ Error Detection:** This testing aims to identify any errors that may occur during the execution of the API. It helps find issues that may not be apparent through static testing.
- **Security testing:** This type assesses the API for security vulnerabilities. It checks for potential threats and ensures that sensitive data is protected from unauthorized access.
- **Validation Testing**: Validation testing ensures that the API meets the business and technical requirements. It checks if the outputs are valid and comply with the expected formats.
- **UI testing:** While primarily focused on the user interface, UI testing checks how the API interacts with the front-end application. It ensures that the user experience is seamless and aligns with the API's functionality.
- **Interoperability Testing**: This type of testing verifies that the API can successfully interact and exchange data with other services and systems. It ensures that the API adheres to industry standards and protocols, allowing seamless integration across different platforms and technologies.
- **WS Compliance Testing**: WS Compliance testing focuses on evaluating the API against the Web Services (WS) standards. This process ensures that the API adheres to relevant specifications, such as SOAP, WSDL, and other related protocols, thus guaranteeing reliability and compatibility within web service ecosystems.
- **Penetration testing:** Using simulated attacks, penetration testing evaluates the API's security by identifying vulnerabilities that could be exploited by hackers.
- **API Hacking**: This involves attempting to access the API in unauthorized ways to discover weaknesses. It helps in understanding potential attack vectors.
- **Fuzz testing:** Fuzz testing sends random or unexpected data to the API to see how it handles unusual input. This is useful for finding bugs and security weaknesses.

Using these different types of API testing helps ensure that the API is reliable, secure, and performs well under various conditions.

Learn more on API Testing Types

6. What are the most important best practices for API testing?

When it comes to API testing, adhering to best practices can significantly enhance the effectiveness and reliability of your testing process. Here are some key principles to consider:

- **Comprehensive Test Coverage:** Ensure that your tests cover all critical endpoints and functionalities of the API, including positive and negative scenarios. This includes testing for various input parameters, error handling, and response codes.
- **Use of Automation:** Leverage automated testing tools to run tests repeatedly and efficiently. Automation helps in identifying regressions quickly as your API evolves, ensuring new changes do not break existing functionality.
- **Documentation Consistency:** Maintain up-to-date API documentation that is clear and detailed. This should include endpoint descriptions, request/response formats, authentication methods, and error codes. Well-documented APIs facilitate better testing and understanding.
- **Environment Management:** Conduct tests in a controlled environment that mirrors the production environment as closely as possible. This includes simulating network conditions and external dependencies to accurately assess the API's behavior under various scenarios.
- **Performance Testing:** Incorporate performance testing to evaluate the API's scalability and reliability under load. Monitor key metrics such as response time, throughput, and resource utilization to ensure the API can handle expected and peak traffic.
- **Security Testing:** Prioritize security testing to identify vulnerabilities and ensure data protection. This includes testing for authentication, authorization, encryption, and common vulnerabilities like SQL injection and cross-site scripting (XSS).

By implementing these best practices, you can enhance the robustness of your API testing efforts, leading to higher quality software and a better user experience.

7. What are the best practices for writing effective API Test Cases

Writing effective API test cases is crucial for ensuring the reliability and functionality of your applications. Here are some best practices to consider:

- **Understand the API Specification**: Before writing test cases, thoroughly review the API documentation to understand the endpoints, parameters, request types, and expected responses.
- **Use Clear and Descriptive Names**: Name your test cases in a way that clearly describes what functionality is being tested. This makes it easier to identify issues during testing and debugging.
- **Test for Edge Cases**: Include tests for both valid and invalid inputs to ensure the API behaves correctly under all circumstances. This helps to identify potential failures before they affect users.
- **Validate All Responses**: Always check the HTTP status codes, response times, headers, and body content to ensure they match the expected outcomes. This helps to verify that the API is functioning as intended.
- **Automate Where Possible**: Implement automated testing tools that can regularly execute your API test cases. Automation can save time and catch regressions in functionality without manual intervention.
- **Keep Tests Isolated**: Create tests that are independent of each other, so that a failure in one test does not affect the others. This isolation helps in accurately pinpointing failures.
- **Document Your Test Cases**: Provide clear documentation for each test case, including the purpose of the test, input values, and expected results. This aids in maintaining the tests over time and helps new team members understand the testing strategy.

By following these best practices, you can create robust and effective API test cases that will enhance your overall testing strategy and contribute to building higher-quality software.

8. What are some best practices for API test automation?

- **Understand the API Specification:** Familiarize yourself with the API documentation to comprehensively understand the endpoints, request methods, and data formats.
- **Use Automation Tools:** Select the right tools for API testing such as Postman, SoapUI, or RestAssured, which can streamline the automation process and improve test reliability.
- **Write Clear and Maintainable Tests:** Ensure that your test scripts are well-structured, easy to read, and maintainable. Use meaningful names for tests and comments where necessary to explain complex logic.
- **Test for Different Scenarios:** Cover a range of test cases including positive, negative, boundary, and edge cases to ensure the API behaves as expected in all situations.
- **Implement Continuous Integration:** Integrate API tests into the CI/CD pipeline to ensure they are executed automatically during builds and deployments, which helps catch issues early.
- **Validate Response Codes and Payloads:** Always check the HTTP response codes as well as the returned data structure and content against expected results to validate the API's functionality.
- **Use Mock Services:** When necessary, utilize mock services to simulate API responses, which can help in cases where the actual service is unavailable or still under development.
- **Perform Security Testing:** Incorporate security testing into your automation suite to identify vulnerabilities in the API, such as authentication issues or data exposure.
- **Monitor API Performance:** Regularly test the performance of the API, including response times and load testing, to ensure it can handle expected levels of traffic without degradation.

9. What steps can you take to plan and implement an effective API testing strategy?

To plan and implement an effective API testing strategy, consider the following steps:

1. **Define Testing Objectives**: Clearly outline what you aim to achieve with your API testing, such as performance validation, functionality verification, or security assurance.
2. **Determine Test Types**: Identify the various types of tests you need, including unit tests, integration tests, functional tests, performance tests, and security tests, to comprehensively evaluate the API.
3. **Select Appropriate Tools**: Choose the right tools and frameworks that align with your testing objectives. Tools like Postman, Swagger, and cURL can facilitate manual and automated testing processes.
4. **Develop Test Cases**: Create detailed test cases that cover a wide range of scenarios, including edge cases, to ensure thorough evaluation of the API's functionality and performance under various conditions.
5. **Set Up a Test Environment**: Establish a proper testing environment that mirrors production as closely as possible to accurately simulate API usage and interactions.

6. **Automate Where Possible**: Automate repetitive test cases to save time and increase efficiency. Continuous integration (CI) processes can help integrate automated tests into the development workflow.
7. **Monitor and Analyze Results**: After executing tests, monitor outcomes carefully. Analyse results to identify trends, defects, or performance issues, which can guide future improvements.
8. **Iterate and Update**: Regularly review and update your API testing strategy based on new findings, incoming feature changes, or shifts in application architecture to ensure it remains effective and relevant.

10. What is dynamic input data, and why is it essential for API testing?

Dynamic input data refers to data that is generated on-the-fly during the testing process rather than being pre-defined or static. This approach is vital for API testing as it enables testers to simulate a diverse range of real-world scenarios, making the testing more robust and comprehensive. By using dynamic data, testers can uncover edge cases that static data may miss, ensuring that the API behaves correctly under various circumstances. Additionally, dynamic input helps to reduce biases introduced by predefined data sets, leading to more accurate and reliable testing outcomes. As APIs often interact with unpredictable data inputs, incorporating dynamic data can significantly enhance the overall effectiveness of the testing process.

11. How do you handle dynamic values in API testing, such as timestamps or random data?

Handling dynamic values in API testing, such as timestamps or random data, requires a strategic approach to ensure accuracy and consistency in test scenarios. One effective method is to use libraries or frameworks that can generate dynamic values programmatically. For instance, timestamps can be derived from the system clock, while random data can be produced using functions provided by programming languages or testing frameworks. Additionally, it is essential to document the generated values during test execution to maintain reproducibility for subsequent test runs. By managing dynamic data effectively, testers can ensure that their API tests are reflective of real-world conditions, thereby improving the reliability of the testing outcomes.

12. Name some of the common protocols used in API Testing?

Some of the protocols using in API Testing are as follows:

- HTTP
- REST
- SOAP
- JMS
- UDDI

13. What are some of the architectural styles for creating a Web API?

Some of the architectural styles for creating web api are as follows.

- Simple URI as the address for the services
- Stateless communication
- HTTP for client-server communication
- XML/JSON as formatting language

14. What is API test environment?

An API test environment refers to a dedicated setup where API testing can be conducted safely and effectively. It contains no Graphical User Interface (GUI). This environment mimics the production environment to provide realistic testing conditions while keeping the production data and systems secure. It typically includes a suitable combination of test servers, databases, and various software tools specifically designed for API testing.

By isolating the testing process, developers and testers can evaluate the API's functionality, performance, and security without risking disruptiveness to live applications. A well-configured API test environment allows for systematic testing, reproducibility of results, and the identification of potential issues before the API is deployed into production.

15. What is an API framework?

An API framework is a set of tools, libraries, and best practices that simplify the process of building and testing APIs. It provides a structured environment that aids developers in creating, managing, and deploying APIs with efficiency and consistency. An effective API framework usually includes functionalities for defining endpoints, handling requests and responses, performing authentication, and facilitating documentation. Additionally, it often integrates testing capabilities, enabling automated testing of APIs to ensure they function correctly and meet specified requirements. By employing an API framework, teams can accelerate development cycles, maintain quality standards, and reduce the risks associated with API integrations.

16. What is an API testing framework?

An API testing framework is a set of guidelines, tools, and practices designed to streamline the process of testing APIs. It provides developers and testers with a structured environment to create, manage, and execute automated test cases that validate the functionality, performance, and security of APIs. These frameworks often support various programming languages and offer features such as test case organization, data-driven testing, and integration with CI/CD pipelines. Examples of popular API testing frameworks include Postman, SoapUI, and Rest-Assured, each catering to different testing needs while enhancing the efficiency and effectiveness of the API testing process.

17. What is an API endpoint?

An API endpoint is essentially a specific URL or URI (Uniform Resource Identifier) that corresponds to a resource or service offered by an API. Each endpoint is designed to handle a particular function or a set of functionalities associated with the API, enabling clients to access, modify, or interact with data. For example, in a weather API, an endpoint might provide current weather data for a specified location. Clients make requests to these endpoints using standard HTTP methods, and the API returns responses, typically formatted in JSON or XML, containing the requested information or the results of an operation. Understanding API endpoints is crucial for effective API integration and usage.

18. How do you handle error responses and exceptions during API Testing?

When testing APIs, it is crucial to effectively handle error responses and exceptions to ensure robust functionality and reliability. First, testers should identify the types of errors that can occur, such as client errors (4xx status codes) and server errors (5xx status codes). During testing, it is important to create test cases that deliberately trigger these errors to verify that the API responds correctly.

For example, when an invalid request is made, the API should return an appropriate error response, which includes a status code and a descriptive message in the response body. This helps the client understand what went wrong. Additionally, testers should check that any exceptions encountered are logged appropriately and that sensitive information is not exposed in error responses. Implementing comprehensive error handling in API testing not only helps to diagnose issues but also enhances the user experience by providing clear feedback on failures.

19. How can you effectively test APIs for performance and scalability?

To effectively test APIs for performance and scalability, several key strategies can be employed.

First, load testing is essential to assess how well the API performs under expected user load, simulating multiple requests to identify potential bottlenecks. Tools like JMeter or LoadRunner can help generate traffic and measure response times.

Second, stress testing goes a step further by pushing the API beyond its limits to determine how it behaves under excessive loads and to pinpoint breaking points.

Additionally, using monitoring tools can provide insights into resource usage, helping to track performance metrics such as latency, throughput, and error rates.

Lastly, it is crucial to conduct scalability testing to evaluate the API's ability to handle increased loads as user demand grows, ensuring that it can effectively scale up without compromising performance.

20. What is Latency in API Testing?

Latency in API testing refers to the time it takes for an API to respond to a request after it has been submitted. It encompasses various delays, including network transmission time, server processing time, and any queuing time that might occur before the API can fulfill the request. Measuring latency is crucial because it directly affects user experience; a high latency can lead to slow application performance and potential user dissatisfaction. Effective API testing should include latency measurements to ensure that the API meets performance requirements and can handle expected load levels efficiently.

21. Difference between API and Web services?

Parameters	API	Web Service
Definition	API is an Application Programming Interface that acts as an interface between two applications.	Web services are a type of API that must be accessed through a network connection.
Protocols Support	It provides support for multiple protocols including HTTP/s, WebSocket, etc.	It primarily provides support for HTTP protocol.
XML Support	API supports both XML and JSON formats.	Web service supports XML exclusively.
Hosting Platform	It can be hosted on various platforms including local and cloud environments.	It is typically hosted on IIS (Internet Information Services).
Usage	It is utilized as an interface for communication between various applications. e.g. DLL files in C/C++, Jar files/ RMI in java, Interrupts in Linux kernel API etc.	It is specifically used for communication protocols such as REST, SOAP, and XML-RPC.
Network Dependency	APIs may function without a network connection, allowing for local interactions and services.	Web services require a network connection for communication, making them dependent on internet availability.

22. What is SOAP?

SOAP, or Simple Object Access Protocol, is a protocol used for exchanging structured information in web services. It relies on XML as its message format and typically operates over HTTP or SMTP. SOAP provides a way for programs running on different operating systems to communicate with each other by sending messages. It is known for its ability to support complex operations and ensure message integrity and security through WS-Security standards.

23. What is REST API?

REST (Representational State Transfer) API is an architectural style that leverages HTTP requests to access and manipulate data. In a RESTful system, resources are identified by URIs, and standard HTTP methods such as GET, POST, PUT, and DELETE are used to perform operations on them. REST APIs are known for their simplicity and scalability, allowing for stateless interactions between clients and servers, which enhances performance and optimizes resource usage. This approach makes REST APIs widely used for building web services that are accessible from different platforms and devices.

24. Difference between SOAP and REST?

SOAP:

1. SOAP is a protocol through which two computers communicate by sharing XML document
2. SOAP supports only XML format
3. SOAP does not support caching
4. SOAP is slower than REST
5. SOAP is like a custom desktop application, closely connected to the server
6. SOAP runs on HTTP but envelopes the message

REST:

1. REST is a service architecture and design for network-based software architecture
2. REST supports different data formats
3. REST supports caching

4. REST is faster than SOAP
5. REST client is just like a browser and uses standard methods An application has to fit inside it
6. REST uses the HTTP headers to hold meta information

25. What are the common tests conducted on APIs?

Some of the common tests we perform on APIs are as follows.

- Verify whether the return value is based on the input condition. The response of the APIs should be verified based on the request.
- Verify whether the system is authenticating the outcome when the API is updating any data structure
- Verify whether the API triggers some other event or request another API
- Verify the behavior of the API when there is no return value

26. What are the advantages of API Testing?

- **Time effective:** API Testing is time effective when compared to GUI Testing. API test automation requires less code so it can provide faster and better test coverage.
- **Reduce cost:** API Testing helps us to reduce the testing cost. With API Testing we can find minor bugs before the GUI Testing. These minor bugs will become bigger during GUI Testing. So finding those bugs in the API Testing will be cost-effective to the Company.
- **Language independent:** API testing is language independent, allowing developers to utilize any programming language to write automated tests by using standard data formats like XML or JSON for communication.
- **Core functionality testing:** API Testing is quite helpful in testing Core Functionality. We can test the APIs without a user interface. In GUI Testing, we need to wait until the application is available to test the core functionalities.
- **Reduce risk:** API Testing helps us to reduce the risks.

27. What are the disadvantages of API Testing?

- **Limited User Interface Testing**: API testing does not cover the user interface, which means potential issues related to usability and visual elements may go unnoticed.
- **Complexity**: Developing and maintaining API tests can be more complex than UI tests, especially if APIs have extensive configurations or require intricate setup.
- **Requires Technical Skills**: API testing typically necessitates a deeper understanding of APIs and programming, which can be a barrier for non-technical team members.
- **Mocking Dependencies**: When testing APIs that depend on other services, creating mock services can add an extra layer of complexity and may not always accurately reflect real-world scenarios.
- **Lack of Visual Feedback**: API testing can be less intuitive than UI testing, as it does not provide the immediate visual feedback of how an application behaves or looks, making it harder to spot front-end issues.

28. What exactly needs to be verified in API Testing?

Basically, on API Testing, we send a request to the API with the known data and we analyze the response.
1. Data accuracy
2. HTTP status codes
3. Response time
4. Error codes in case API return any errors
5. Authorization checks
6. Non-functional testing such as performance testing, security testing

29. Name some tools used for API Testing?

Some of the tools used for API Testing are as follows:

- **Postman:** Widely used for manual and automated testing, Postman offers a user-friendly interface to create, send, and test API requests.

- **SoapUI:** Ideal for testing SOAP and REST APIs, SoapUI provides comprehensive testing features including functional testing, load testing, and security testing.
- **RestAssured:** A Java-based library that simplifies the testing of REST services, RestAssured allows for writing readable and maintainable tests using a fluent API.
- **JMeter:** Primarily a performance testing tool, JMeter can also be utilized for API testing, enabling the simulation of various load scenarios and testing the API's performance under stress.
- **Swagger:** With tools like Swagger UI and Swagger Codegen, developers can not only document APIs but also test and generate client code for different programming languages.
- **Karate:** A test automation framework that combines API testing, UI testing, and performance testing in a single solution, simplifying the testing process with its DSL (Domain Specific Language).
- **Katalon Studio:** An all-in-one automation tool that supports API, Web, and Mobile testing, providing features for easy API test creation and execution.

Some other tools are Assertible, Tricentis Tosca, Apigee, JMeter, API Fortress, Parasoft, HP QTP(UFT), vREST, Airborne, API Science, APIary Inspector, Citrus Framework, HttpMaster Express, Mockbin, Ping API, Pyresttest, Rest Console, RoboHydra Server, SOAP Sonar, Unirest, WebInject

> ***Learn more on** API Testing Tools*

30. What is API Documentation in API testing?

API documentation is a user manual for a software interface that allows different programs to communicate with each other. It explains how to use the API, detailing its features and functionalities. This documentation typically includes details about API endpoints, request and response formats, authentication and authorization methods, error handling, and other relevant information. It serves as a guide for developers, helping them understand how to connect their applications with the API effectively. Good API documentation is clear, easy to follow, and provides examples to illustrate how the API works.

31. What are the important factors should be taken into account while writing API document?

When creating API documentation, clarity and comprehensiveness are paramount. First, ensure that the language used is clear and concise, avoiding technical jargon that may confuse users. Including comprehensive reference documentation is essential, as it provides context and details that users need to understand the API's capabilities fully. Additionally, it's vital to incorporate documentation into the overall development process; this practice helps keep the documentation up-to-date with any changes or improvements made to the API. Quickstart guides should also be provided to help users get up and running quickly, offering practical examples and simplified instructions. Lastly, consider the target audience's experience level, tailoring the documentation to meet both novice and advanced user needs.

32. List some commonly used API documentation templates?

Some of the API documentation templates are as follows.

- Swagger
- FlatDoc
- RestDoc
- API blueprint
- Slate
- Miredot
- Web service API Specification.

33. Name some of the API examples which are quite popular.

Some of the popular API examples are

- Google Maps API

- YouTube
- Twitter
- Amazon Advertising API

34. Difference between API testing and Unit Testing?

Criteria	API Testing	Unit Testing
Purpose	Verifies the functionality, reliability, and security of the API as a whole.	Tests individual components or functions of the application.
Execution	Can be executed by testers or developers, often involves multiple systems.	Primarily executed by developers, usually in isolation.
Focus	Emphasizes on API contracts and integration with external systems.	Focuses on the internals of the code, checking logic and functionality.
Level	Performed at the interface level, where the API is exposed.	Conducted at the unit level, typically by developers during the coding phase.
Tools	Utilizes tools like Postman, SoapUI, or REST Assured.	Commonly uses frameworks like JUnit, NUnit, or pytest.
Scope	Tests the interaction between different components and their responses.	Tests specific functions, methods, or procedures in isolation.
Source Code	Not involved	Involved
Testing Type	API testing is a form of Black box testing	Unit testing is a form of White box testing
Testing Phase	API testing is conducted after the build is ready for testing.	Unit testing is conducted prior to the process of including the code in the build.

35. What are the main challenges faced in API testing?

Some of the challenges we face while doing API testing are as follows

- Selecting proper parameters and its combinations
- Categorizing the parameters properly
- Proper call sequencing is required as this may lead to inadequate coverage in testing
- Verifying and validating the output
- Due to the absence of GUI, it is quite difficult to provide input values
- Lack of proper documentation

36. What are the types of bugs we face when performing API testing?

Issues observed when performing API testing are

- Failure to handle negative test cases
- Duplicate or missing API functionality
- Incompatible error handling mechanism
- Stress, performance, and security issues
- Reliability issues
- Improper status codes
- Improper error responses
- Multi-threaded issues
- Improper errors
- Unimplemented errors
- Unused flags

37. How is UI testing not similar to API testing?

UI (User Interface) testing is to test the graphical interface part of the application. Its main focus is to test the look and feel of an application. On the other hand, API testing enables the communication between two different software systems. Its main focus is in the business layer of the application.

38. Name some most commonly used HTTP methods?

Some of the HTTP methods are

GET: It enables you to retrieve data from a server
POST: It enables you to add data to an existing file or resource in a server
PUT: It lets you replace an existing file or resource in a server
DELETE: It lets you delete data from a server
PATCH: It is used to apply partial modifications to a resource
OPTIONS: It is used to describe the communication options for the target resource
HEAD: It asks for a response identical to that of a GET request, but without the response body

39. Can you use GET request instead of PUT to create a resource?

No, GET request only allows read only rights. It enables you to retrieve data from a server but not create a resource. PUT or POST methods should be used to create a resource.

40. What is the difference between PUT and POST methods?

PUT and POST methods are sometimes confused in regards to when each should be used. Using POST request, our intent is to create a new object on the server whereas with PUT request, our intent is to replace an object by another object.

POST should be used when the client sends the page to the server and then the server lets the client know where it put it. PUT should be used when the client specifies the location of the page

41. Is it possible to use a GET request instead of a PUT request to create a resource?

No, it is not appropriate to use a GET request to create a resource. According to the principles of RESTful architecture, a GET request is designed to retrieve data from a server without causing any side effects or changes to the resource state. In contrast, a PUT request is specifically intended for creating or updating resources on the server.

42. What is API versioning, and why is it crucial for effective API testing?

API versioning is all about including a version number in the API endpoint or headers. This helps keep things compatible as the API evolves. It’s super important in API testing too—you want to make sure you’re testing the right version and that any updates won’t mess up existing client applications.

43. What are HTTP Status Codes? Name some of the important Status Codes?

HTTP Status Codes are standardized codes returned by a server in response to a client’s request, indicating the outcome of the request. They are grouped into categories such as informational responses, successful responses, redirection messages, client errors, and server errors.

Here are the primary categories and some common codes:

- **1xx: Informational**
  - **100 Continue:** The initial part of a request has been received and has not yet been rejected by the server.
  - **101 Switching Protocols:** The server is switching protocols as requested by the client.
- **2xx: Success**
  - **200 OK:** The request has succeeded, and the server has returned the requested data (or acknowledgement for methods like POST).
  - **201 Created:** The request has been fulfilled, resulting in the creation of a new resource.

    - **204 No Content:** The server successfully processed the request, but is not returning any content.
- **3xx: Redirection**
    - **301 Moved Permanently:** The requested resource has been assigned a new permanent URI.
    - **302 Found:** The resource is temporarily located at a different URI, indicated in the response.
    - **304 Not Modified:** The resource has not been modified since the last request, allowing the cached version to be used.
- **4xx: Client Error**
    - **400 Bad Request:** The server cannot process the request due to a client error (e.g., malformed request syntax).
    - **401 Unauthorized:** Authentication is required and has failed or has not yet been provided.
    - **404 Not Found:** The server cannot find the requested resource, indicating that it may not exist.
- **5xx: Server Error**
    - **500 Internal Server Error:** The server encountered an unexpected condition that prevented it from fulfilling the request.
    - **502 Bad Gateway:** The server, while acting as a gateway or proxy, received an invalid response from the upstream server.
    - **503 Service Unavailable:** The server is currently unable to handle the request due to temporary overload or maintenance of the server.

44. What is the purpose of the request and response body in API testing

The request body in API testing serves to send data to the server, allowing for the execution of operations such as creating or updating resources. Conversely, the response body contains the data returned by the server, which provides insights into the outcome of the request and includes any relevant information or error messages related to the operation performed.

45. What is the purpose of the request and response body in API testing

The request headers in API testing provide essential metadata that informs the server about the nature of the request, including content type and authentication credentials. In contrast, the response headers convey crucial information about the server's response, such as the status code and any caching policies. Together, these headers facilitate effective communication between the client and server, ensuring that the API behaves as expected.

46. What is API Mocking and Its Purpose in API Testing?

API mocking refers to the practice of creating a simulated version of an API that mimics the behavior of a real API. This technique is commonly employed in API testing to enable developers and testers to evaluate the functionality of applications without relying on the actual API. By using mock APIs, teams can conduct tests in a controlled environment, ensuring that the application performs as expected, even in scenarios where the real API may be unavailable or unpredictable. This approach not only accelerates the development process but also enhances the reliability of testing outcomes.

47. What are the advantages of using API mocking in API Testing?

- **Early Testing**: Allows testing to begin even before the actual API is available.
- **Independent Development**: Developers and testers can work separately, reducing dependencies on backend services.
- **Cost-Effective**: Saves time and resources by eliminating the need for a fully developed API during testing.
- **Improved Reliability**: Testing can be conducted without relying on the availability of external services, leading to more consistent results.
- **Faster Feedback**: Quickly identifies issues, enabling faster iterations and improvements in the development cycle.
- **Simulated Error Handling**: Testers can easily simulate different responses (including errors) from the API, allowing for robust error handling tests.
- **Documentation and Training Aid**: Serves as a useful resource for team members to understand API functionality and for onboarding new developers.

48. What is the difference between authentication and authorization?

Authentication and authorization are two fundamental concepts in security. Authentication is the process of verifying the identity of a user or system, ensuring that the person or entity is who they claim to be. In contrast, authorization determines what an authenticated user is allowed to do, defining their access levels and permissions within the system.

49. How do you perform load testing on APIs?

Load testing on APIs involves simulating a large number of concurrent users or requests to evaluate how the API performs under stress. This is typically accomplished using load testing tools that can generate and send a high volume of requests to the API, allowing testers to measure response times, error rates, and overall stability. The results help identify bottlenecks, ensuring that the API can handle anticipated traffic demands efficiently.

50. What is API security testing, and why is it important?

API security testing is the process of evaluating an API to identify vulnerabilities and ensure that data is protected from unauthorized access or manipulation. It is crucial because APIs often serve as gateways to sensitive data and functionalities, making them prime targets for cyberattacks. Ensuring robust security measures in API design and testing helps maintain user trust and safeguard critical systems.

51. What are some common security vulnerabilities in APIs?

APIs are susceptible to several common security vulnerabilities, which can lead to significant risks if not properly addressed.

One major threat is injection attacks, where an attacker can insert malicious code into an API request, potentially leading to data breaches or unauthorized command execution.

Additionally, vulnerabilities related to authentication and authorisation can arise when APIs do not implement proper access controls, allowing attackers to gain access to sensitive data or functionalities without proper credentials.

Another one, cross-site scripting (XSS) attacks can occur when APIs do not properly sanitize user inputs, enabling attackers to execute scripts in the context of users' browsers, which can compromise user sessions or steal sensitive information.

Addressing these vulnerabilities is essential to ensure the security and integrity of API interactions.

52. What is Cross-Site Scripting (XSS), and How Can It Be Prevented in API Testing?

Cross-Site Scripting (XSS) is a vulnerability that allows attackers to inject malicious scripts into web applications, which are then executed by unsuspecting users' browsers. This can lead to various issues, such as session hijacking, data theft, and defacing of web pages. In the context of APIs, XSS attacks can occur when user input is not properly sanitized, allowing harmful scripts to be processed and returned in API responses.

To prevent XSS in API testing, it is crucial to implement several key practices:

1. **Input Sanitization**: All user inputs should be thoroughly validated and sanitized, stripping away any potentially harmful scripts or elements before processing them in the API.
2. **Output Encoding**: Ensure that any data returned to the client is properly encoded, so that any potentially executable code is treated as plain text rather than as a script.
3. **Content Security Policy (CSP)**: Implement CSP headers to limit the sources from which scripts can be executed, thereby reducing the risk of XSS attacks.
4. **Regular Security Testing**: Conduct regular security assessments and utilize automated testing tools to identify potential XSS vulnerabilities within the API.

By following these preventive measures, developers can significantly mitigate the risks associated with XSS in API interactions.

53. What is Cross-Site Request Forgery (CSRF), and How Can It Be Prevented in API Testing?

Cross-Site Request Forgery (CSRF) is a type of attack that tricks a user's browser into making unwanted requests to a different site where the user is authenticated. This can lead to unauthorized actions being performed on behalf of the user, such as changing account settings or submitting data without their consent. In the context of APIs, CSRF attacks are particularly concerning because they can exploit authenticated sessions to execute unauthorized commands.

To prevent CSRF in API testing, developers should adopt several important strategies:

1. **CSRF Tokens**: Implement anti-CSRF tokens that are unique per session and are required for every state-changing request. This ensures that the request is intentional and originates from a valid user action.
2. **SameSite Cookie Attribute**: Use the SameSite attribute for cookies to restrict how cookies are sent with cross-site requests, providing an additional layer of security against CSRF attacks.
3. **Check HTTP Referer Header**: Validate the HTTP Referer header to ensure that requests are coming from the expected origin, rejecting any that originate from untrusted sites.
4. **User Confirmation for Sensitive Actions**: Require additional user confirmations for sensitive actions, such as submitting a form or changing account settings, to safeguard against unintended submissions.

By implementing these protective measures, developers can significantly reduce the risk of CSRF attacks in their API systems.

54. What is API performance testing, and How do you measure the performance of APIs in your tests?

API performance testing is the process of evaluating the speed, scalability, and stability of an API under various loads and conditions. It ensures that the API can handle a specific number of calls without degrading user experience or causing failures. To measure the performance of APIs during tests, developers often use metrics such as response time, throughput, and error rates, along with tools like JMeter or Postman to simulate different levels of traffic and analyze the results.

55. What is API Virtualization and What are the Benefits of Using API Virtualization in API Testing?

API virtualization is the technique of simulating the behavior of APIs that are not yet developed or are unavailable for testing purposes. By creating a virtual model of an API, developers and testers can conduct testing on applications that rely on those APIs without having to wait for the actual service to be ready. This approach allows for more efficient testing cycles and helps teams to identify issues earlier in the development process.

The benefits of using API virtualization in API testing include:

1. **Accelerated Testing**: Testers can work in parallel with development teams without waiting for the actual APIs, speeding up the overall development process.
2. **Cost Efficiency**: Virtualized APIs reduce the need for costly infrastructure and external services that may be required for testing, leading to decreased operational costs.
3. **Isolation of Systems**: Virtualization allows teams to test API interactions in a controlled environment without affecting the production systems or relying on external systems that may be unstable.
4. **Improved Test Coverage**: With the ability to simulate various scenarios, including edge cases, teams can ensure that their applications are robust and can handle unexpected inputs gracefully.
5. **Enhanced Collaboration**: Teams can work more effectively together by using shared virtual APIs, leading to better communication and workflows among development, testing, and operations teams.

By leveraging API virtualization, teams can create a more agile and responsive testing environment, ultimately leading to higher-quality software products.

Don’t Miss:

- 30+ Postman Interview Questions
- Learn API Testing in 10 Minutes

I would like to conclude this post “API Testing Interview Questions” here. Final words, Bookmark this post “API Interview Questions And Answers” for future reference.
Here I have hand-picked a few posts which will help you to learn more interview related stuff:

- Postman Tutorial (Beginner to Advanced Level)