**API testing interview Questions.**

**What is API Testing?**

**A**pplication **P**rogramming **I**nterface.

API testing, or Application Programming Interface testing, is a type of software testing that focuses on verifying and validating the functionality, performance, security, and reliability of APIs and the core business logic to ensure that different software applications to communicate with each other as per the requirements. API testing is crucial in the development lifecycle as it helps identify issues early, promotes consistency across different components, and ensures that APIs meet their functional and non-functional requirements.

**Functional API Testing:**

Request and Response: Testing the requests sent to the API and the responses it returns.

Data Integrity: Verifying that data is transmitted correctly between the client and server.

**Error Handling**: Ensuring that the API handles errors gracefully and returns appropriate error codes and messages.

**Performance Testing**:

**Scalability**: Checking if the API can handle an increase in the number of requests.

**Authorization and Authentication**: Verifying that the API requires proper authentication and authorization before granting access to resources.

Data Encryption: Ensuring that data transmitted over the API is encrypted to protect sensitive information.

**Access Control**: Confirming that users can only access the resources they are authorized to use.

Reliability Testing:

**Consistency**: Ensuring the API behaves consistently across different environments and under various conditions.

**Documentation**: Checking the clarity and completeness of API documentation.

**Compatibility**: Verifying that the API works seamlessly with different clients, servers, and environments.

Regression Testing:

**Mocking**: Creating simulated versions of the API for testing purposes, especially when the real API is not available.

Virtualization: Testing the API in a controlled virtual environment to simulate real-world conditions.

**APR response code.**

Testing the API (application programming interface) which the developer has created. Testing the core business logic of the application. Like certain data is sent and how they are behaving and make sure the response is correct and behaving of the application is correct.

Get/HTTP 200 status is whatever request we make should be successful.

Get/HTTP 201 status whatever request you made; some data will be created on requested server.

Get/HTTP 204 status, sent request without expect to get respond. Use for Delete

Get/HTTP 300 status. Remove permanently.

Get/HTTP 400. BAD request invalid or user error.

Get/HTTP 401. Unauthorized

Error ON API- Wrong header, failure to handle negative request, unused flag, Unimplemented error, Improper status code.

**HTTP Status Codes for RESTful APIs:**

100 Continue: The initial part of the request was received, and the client should proceed with the request.

200 - Success:

200 OK: The request was successful.

201 Created: The request has been fulfilled and resulted in a new resource being created.

204 No Content: The request was successful, but there is no additional content to send in the response payload.

3xx - Redirection:

301 Moved Permanently: The requested resource has been permanently moved to another location.

304 Not Modified: The resource has not been modified since the last request.

4xx - Client Errors:

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

401 Unauthorized: Similar to 403, but authentication is required and has failed or has not been provided.

403 Forbidden: The client does not have access rights to the content, so the server is refusing the request.

404 Not Found: The server can't find the requested resource.

5xx - Server Errors:

500 Internal Server Error: A generic error message returned when an unexpected condition was encountered on the server.

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 not ready to handle the request.

HTTP Status Codes for SOAP APIs:

SOAP APIs typically use HTTP status codes, but the specific interpretation of these codes can depend on the context of the SOAP envelope. Common HTTP status codes are still relevant, but SOAP itself uses its own set of status codes within the SOAP envelope.

200 OK: The request was successful.

500 Internal Server Error: A generic error message returned when an unexpected condition was encountered on the server.

503 Service Unavailable: The server is not ready to handle the request.

Differences:

Representation:

REST: Typically uses JSON or XML for data representation.

SOAP: Primarily uses XML for message format.

Protocol:

REST: Often relies on HTTP for communication.

SOAP: Can use various protocols, including HTTP, SMTP, and more.

Statelessness:

REST: Stateless by nature. Each request from a client contains all the information needed.

SOAP: Can be stateful or stateless, depending on implementation.

Ease of Use:

REST: Generally considered more straightforward and easier to implement.

SOAP: Can be more complex due to its XML-based structure.

Flexibility:

REST: More flexible, supporting various data formats.

SOAP: Typically rigid in its XML format.

Standards:

REST: No official standards; relies on conventions.

SOAP: Has a set of standardized protocols (WSDL, UDDI, etc.).

REST, or Representational State Transfer, is an architectural style for designing networked applications. RESTful APIs (Application Programming Interfaces) adhere to the principles of REST. These APIs are designed to allow different software systems to communicate over standard web protocols. Key characteristics of RESTful APIs include:

Statelessness: Each request from a client to a server must contain all the information needed to understand and process the request. The server doesn't store any information about the client between requests.

Resource-Based: Resources (data or services) are identified by URIs (Uniform Resource Identifiers) and manipulated using standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.

Representation: Resources can be represented in various formats, such as JSON or XML. Clients interact with resources through these representations.

Client-Server Architecture: The client and server are separate entities that communicate over a network. The client is responsible for the user interface, and the server is responsible for processing requests and managing resources.

HTTP Methods in REST:

GET: Retrieve data from the specified resource.

POST: Submit data to be processed to a specified resource.

PUT: Update a resource or create a new resource if it does not exist.

PATCH: Apply partial modifications to a resource.

DELETE: Delete the specified resource.

SOAP API:

SOAP, or Simple Object Access Protocol, is a protocol for exchanging structured information in web services. Unlike REST, SOAP is a protocol with a specific set of standards. SOAP messages are XML-based and usually sent via HTTP, SMTP, or other transport protocols. Key characteristics of SOAP include:

Envelope Structure: SOAP messages are enveloped in a structure that includes a header and a body.

XML Format: The content of the SOAP message is typically in XML format.

Protocol Independence: SOAP can operate over various protocols, not limited to HTTP.

Stateful or Stateless: SOAP can be implemented in a stateful or stateless manner, depending on the application's requirements.

Standardization: SOAP has a set of standardized protocols, including WSDL (Web Services Description Language) for describing the functionality offered by a web service.

**1.**     **How do you test an API?**

Answer: Testing an API typically involves creating API requests to send to a server, usually in the form of a URL, and then validating the response that is received. This can include checking for the correct format and structure of the response, the status code, and the data returned. Additionally, it is important to also test for any security vulnerabilities that may exist in the API.

**2. What techniques do you use to test an API?**

1. Unit Testing - This involves testing individual units of code in isolation, such as functions, classes, and modules.

2. Integration Testing - This involves testing how multiple parts of the code interact with each other, as well as how the API interacts with external services.

3. Functional Testing - This involves testing the API for its intended functions, such as verifying the response codes, response times, and data accuracy.

4. Load Testing - This involves testing the API under heavy load to identify any potential bottlenecks or performance issues.

5. Security Testing - This involves testing the API for security vulnerabilities, such as SQL injection, cross-site scripting, and authentication flaws.

**3. What tools do you use to test an API?**

Postman

**4. What is a “positive” test case for an API?**

Answer: A positive test case for an API is a test that validates the expected behavior of the API when given valid input. This includes testing for successful responses, proper data formatting, and any other expected results. For example, a positive test case for an API that retrieves user data might include providing valid user credentials and verifying that the returned response contains the expected user information.

**5. What is a “negative” test case for an API?**

Answer: A negative test case for an API test the functionality of the API when incorrect or invalid input is provided. This test case is designed to ensure that the API handles unexpected input gracefully and does not cause errors or unexpected behavior.

**6. How do you ensure that an API is secure?**

Answer: There are several measures that can be taken to ensure that an API is secure. These include: 1. Implementing authentication and authorization protocols to limit access to the API. 2. Using HTTPS and TLS to encrypt data in transit. 3. Using API keys and tokens to limit usage. 4. Implementing rate limiting to prevent malicious actors from flooding the API with requests. 5. Implementing input validation to reduce the risk of malicious input. 6. Implementing logging and monitoring to identify and resolve issues quickly. 7. Applying security patches and updates to keep the API up to date.

**7. How do you identify potential vulnerabilities in an API?**

Answer: Potential vulnerabilities in an API can be identified by performing a security assessment, which includes looking for potential weak points in the design and structure of the API, such as authentication weaknesses, authorization weaknesses, input validation issues, parameter manipulation issues, and broken access control. Additionally, one can run a security audit to uncover any potential vulnerabilities in the code, such as buffer overflows, SQL injection, and cross-site scripting (XSS). Finally, penetration testing can be used to identify any potential weaknesses and vulnerabilities in the API.

**8. How do you handle API changes during testing?**

Answer: When testing an API, it is important to be prepared for changes. If the changes are minor, they can be tested manually. If the changes are more significant, automated tests should be written to ensure the API is functioning properly with the new changes. Additionally, regression tests should be written to make sure the new changes do not adversely affect existing functionality. Finally, tests should be written to make sure that the changes are working as expected.

**9. How do you debug an API when you encounter a problem?**

Answer: When debugging an API, it’s important to first identify the problem. After that, the best way to debug an API is to use a tool like Postman or Curl to make API requests and analyze the response. Additionally, examining the API’s logs, checking for any errors in the code, and running tests on the API can also be helpful in debugging.

**10. What experience do you have with REST APIs?**

Successfully conducted comprehensive testing of REST APIs associated with a medical device. This involved a multifaceted approach, including functional testing where I meticulously crafted test cases to validate the accuracy and reliability of the API across various endpoints, ensuring it met the critical functionality requirements of the medical device. Security testing was a top priority, and I implemented scenarios to rigorously validate authentication and authorization mechanisms, safeguarding sensitive medical data.

**11- How do you test an API in detail?**

Answer- To test an API in detail, you should first create both positive and negative test cases that cover all the possible scenarios. Once the test cases are ready, you should follow these steps: 1. Send a request to the API with valid parameters and check the response. 2. Check if the response code is valid. 3. Check if the response data is as expected. 4. Test each API endpoint to make sure it is working properly. 5. Test the API for security vulnerabilities. 6. Test the API for performance and scalability. 7. Test the API documentation for accuracy and completeness. 8. 10. Monitor the API for any changes in the response data.

Some of the commons API issue and how to overcome them.

**Scenario: Unauthorized Access**

1. Issue: Users can access certain API endpoints without proper authentication.

* **Resolution**: Implement proper authentication mechanisms such as API keys, OAuth tokens, or JWTs. Ensure that each request to sensitive endpoints includes valid authentication credentials.
* **Testing**: Create test cases to verify that unauthorized requests are appropriately rejected with a 401-status code. Test valid authentication scenarios to ensure that authorized requests are processed correctly.

**Scenario: Incorrect Data Format**

1. Issue: The API is not handling data in the expected format, leading to parsing errors.

* **Resolution**: Validate incoming data against the expected format and provide clear error messages for data that doesn't meet the requirements.
* **Testing**: Design test cases with malformed or incorrectly formatted data. Verify that the API responds with appropriate error codes (e.g., 400 Bad Request) and meaningful error messages.

**Scenario: Slow Response Times**

1. Issue: The API response times are slower than expected, impacting performance.

* **Resolution**: Optimize the API code, database queries, or network calls to improve response times. Consider caching or implementing load balancing.
* **Testing**: Conduct performance testing under various load conditions to identify response time bottlenecks. Ensure that the API meets performance requirements and responds within acceptable time limits.

**Scenario: Missing or Incomplete Data**

1. Issue: The API is not returning all the expected data or is missing certain fields.

* **Resolution**: Review the API logic to ensure that it fetches and returns all necessary data. Update documentation to reflect the expected response structure.
* **Testing**: Design test cases to validate that all expected data is present in the API responses. Include scenarios where certain fields might be null or empty.

Scenario: Inconsistent Error Handling

1. Issue: The API does not consistently handle errors or provides inconsistent error messages.

* **Resolution**: Implement a standardized error handling mechanism across all API endpoints. Use HTTP status codes and provide clear, consistent error messages.
* **Testing**: Create test cases for different error scenarios (e.g., invalid input, server errors) and verify that the API consistently returns appropriate HTTP status codes and error messages.

Scenario: Security Vulnerabilities

1. Issue: The API has security vulnerabilities such as SQL injection or insufficient access controls.

* **Resolution**: Implement secure coding practices, input validation, and proper access controls. Regularly perform security audits.
* **Testing**: Conduct security testing to identify and address vulnerabilities. Test common security exploits, such as injection attacks, and verify that the API is resilient to these threats.

1. **Can you describe a scenario where you had to troubleshoot a particularly difficult issue in testing? How did you solve it?**
2. Answer: In my previous role, we were testing a web application that had a complex workflow involving multiple screens and user interactions. We noticed that the application was intermittently freezing during the workflow, making it difficult for users to complete the task. We tried various methods to reproduce the issue, such as changing the input data and using different browsers, but it was still hard to replicate the problem. We eventually used a combination of browser developer tools and server logs to identify that the issue was related to a particular API call that was taking too long to complete. We then worked with the development team to optimize the API call and resolve the issue.