Click to prove you're human

Share copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. ServicesTravel ExpertiseInsightsCompanyEvery application you build today relies on APIs. That means its crucial to thoroughly verify APIs before rolling out the product to the client or end-users. Some QAs, however, still ignore this layer of the test pyramid and proceed right to UI testing at their peril. In this article, we explain why you should do API testing and how to approach it.We have already written in-depth on what an API is. Weve also elaborated on the basics of software quality management. This time we connect the dots under the topic of API testing.Read our dedicated article to learn aboutWebSockets, a communication protocol for creating real-time applications.What is API testing?API testing is a set of quality assurance actions that include sending calls to the API, getting output, and validating the system's response against the defined input parameters, in particular, the accuracy of data and data's format, HTTP status codes, and error codes.Usually, API testing is performed on APIs produced by the in-house development team. We dont test third-party APIs, but we can test the way our software accepts their requests. The approach to API testing largely depends on the API type. There are web APIs aka web services, database APIs that connect applications with DB management systems, operating systems APIs, and remote APIs for accessing resources located outside the device requesting them.In this article, well focus on in-house web APIs and their quality assurance.API in terms of the three-tier architectureClient, server, and database are the three independent tiers of software architecture. Usually, a client is a web browser or a mobile application that presents the requested info, while business logic is realized on the server side. Client and server communicate through requests based on different transfer protocols. The system and logic of protocol-dependent requests are the API itself. The role of API in the three-tier architectureIn the most popular software architectural style for data transfers - REST - clients make HTTP/HTTPS requests to the server. The server retrieves the requested data from the database, transforms it according to the business logic, and returns it to the client as a response in JSON, XML, or other formats. Finally, the UI displays it to the user.Another type of web API is SOAP API - a legacy web communication protocol that is still in use. SOAP isnt limited to HTTP/HTTPS protocol but supports many others including TCP, SMTP, and FTP, but it works with XML format only. You can find out more about SOAP and how it differs from REST in our dedicated article.And now, lets see how it works from the QA side.Why API testing is importantA basic concept in automated testing is a test pyramid. It demonstrates how to tackle automated testing for a project: what parts to test first as they form the base of the pyramid, and what to leave for the final stages - the pyramids tip.Following the pattern of the testing pyramid, the first is the unit test layer - code testing often performed by developers while writing it. Then comes the server layer where API testing belongs. Later, when the front end is finished, UI testing takes place.Three layers of the test automation pyramidAs we already mentioned, API testing is often overlooked. But why is it so important?UI tests cant cover every aspect of back-end testing and turn out to be insufficient for verifying functional paths and API services. This could leave you with bugs rooted in the server or even at the unit level. But now, the cost of mistakes is way higher as it may require rewriting a considerable amount of code and derail the release.The solution here is to start testing at the early stages. If the request doesnt return the needed value at the server layer, it wont be displayed at the UI layer either. Not relying on UI to be ready, API tests can be performed early in the development cycle. This way, theyll kill at least half of the bugs before they get serious.In the Agile processes, where instant feedback is needed, unit and API testing are preferred over GUI testing as they are quicker to create and easier to maintain. While the latter requires considerable rework to keep pace with frequent change.But dont think we dismiss UI testing. Certain defects can only be caught by GUI tests and its the only test type that truly checks the user experience. We just encourage you to pay attention to API testing as well and apply it to smooth the overall development workflow.Types of API testing: What they focus onQA engineers approach API testing from different angles. They test whether it returns correct outputs in the expected format, whether it delivers responses within an acceptable timeframe, and how well it integrates with presentation layer software. Testers check an APIs reactions to edge cases (e.g. failures, unexpected or extreme inputs) and potential security attacks. So lets address each of these aspects.Functional API testsFunctional testing is the assessment of specific functions within the codebase. It makes sure the API actually works within the expected parameters, namely:returns the desired output for a given input; andhandles errors when the results are outside of the expected parameters.One of the functional testing types is Positive / Negative testing. Negative testing checks how an API responds to every possible kind of wrong input, while positive testing verifies the correct functioning of the API when the input conforms to the norm. If positive test cases fail, its a bad sign, as it means the application cant perform even under ideal conditions.API performance testsTo ensure the API can handle the expected or higher load, QA engineers validate its functionality and performance by artificially creating or simulating API calls. We'll outline the types of API performance testing.Load testing. The point of load testing is to measure where the limit of system performance under high load lies. Thats why we measure response times, throughput, server conditions, etc., while increasing the number of calls.Soak testing. Load tests that run over a long period of time can reveal system instabilities like API memory leaks. So when you have a weekend ahead, leave automated soak tests running. On Monday, it will show you whether any unwanted behavior has emerged.Stress testing. The idea is to gradually increase the count of virtual users to find the point at which the API starts throwing errors, slows down, or stops responding.Spike testing. Unlikestress testing, here an API undergoes a sudden and extreme increase or decrease in the number of users over a short period of time. Spike testing checks whether the API is able to stabilize and return to normal functioning after that.Scalability testing. You want to be sure that your system performance scales according to the changing load. To do so, increase the number of incoming requests and monitor whether it causes a proportional increase in response time.API security testsSecurity, penetration, and fuzz testing are the components of the security auditing process aimed at testing an API for vulnerabilities from external threats.Security testing. It validates whether security requirements are met. This includes authentication, permissions, and access controls, namely:what type of authentication is required to use the API;how well sensitive data is encrypted; andwhat authorization checks are set for resource access, etc.Penetration testing. Taking security testing a step further, in penetration testing, certain API functions, resources, processes, or the entire API is under attack from the outside. This determines whether the threat vector can be reached.Fuzz testing. The last step in the security audit tests the API at its absolute limits. Forcibly inputting massive amounts of random data, it tests whether the API will stand it or end up with negative behavior like a forced crash or overflow.API integration and reliability testsBeing at the center of integrations between internal or third-party services, APIs need to pass the following tests.Integration testing. Focused on communication between APIs, it ensures they are well-connected and dont cause bugs in other APIs.Integration API testing, Source: SoapUIReliability testing. When synced with various devices, an API should be tested for any possible disconnections.How to go about testing the APIAPI testing is performed by submitting requests to the software through the API and then checking if it returns the expected data. When automated, API testing can be easily performed on a regular basis. The main advantage of automation is speed - less lag time between development and QA and less time spent on debugging in production.QA engineers perform automated API testing using a testing tool that drives the API. We will discuss the most popular solutions in the next section.API testing step by step, Source: Capital TerefeStep 1: Create API testing requirements. To be able to plan API tests, first, you need to determine testing boundaries and requirements which include:the purpose of the API and its target consumer,the workflow of the application,the features and functions of the API,other APIs this API interacts with,aspects of testing,the problems were testing for,priorities in testing,the desired output, andthe definition of Pass and Fail.Step 2: Establish the API test environment. When you determined the functional scope of your API, the next step is setting up an API test environment which is usually the task of DevOps engineers. It requires the configuration of the servers, databases, and every resource the API interacts with, depending on the software requirements.Step 3: Make a trial API call. Before diving into thorough testing, make an API call just to check that nothing is broken and the API is operational.Step 4: Define the input parameters. Plan all possible input combinations. Youre going to use them in your test cases, and later - for the authentication of the results to determine whether the API performs as expected.Step 5: Create API test cases. After all the preparations are done, you can write and execute test cases, and, in the end, compare actual results with the expected ones. A good practice is to group them by test category. Examples of API test cases are:testing value in response to the input condition: You define the input and authenticate the output, which can be any type of data or status (e.g. Pass or Fail);checking the behavior of the API in the system when there is no return value in response;tracking events or interruptions triggered by the output;authenticating the effect on the system after data structure updates; andvalidating resources modified by the API call. API testing isnt very different from any other testing. But, it requires specific tools, dedicated to automating it. Among other things, those instruments vary in the programming language theyre based on.Using an API testing tool, QAs usually either take advantage of out-of-the-box solutions or develop a customized framework from its components.A brief comparison of popular API testing toolsPostman API testing: REST API checks using JavaScriptPostman is currently one of the most popular tools used in API testing. Started as a browser extension for API validation, today with integrated test automation features, Postman is much more than just an HTTP client.You can test APIs manually using Postman even with close to zero JavaScript skills. Igor Pavlenko, a Senior QA Engineer at AltexSoft, who is an avid Postman user, claims: The tool requires only limited knowledge of JavaScript. So, basically, any Manual QA can create automated API tests just by using Postman predefined snippets.For automated QAs with extensive knowledge of JavaScript, Postman has a JS testing library.Verifying the status code to be 201 via PostmanPostman advantages. The tool is praised for ease of use and speed, tons of features, and many integrations. It has a friendly UI for constructing requests and reading responses, allowing automated tests to be created quickly. Postman runs on local machines so that you can stay in control of your data. Using its command-line tool Newman, you can integrate these tests in continuous integration (CI) environments.However, Igor also warns that its hard to maintain extensive API collections and its built-in versioning system is weak, so, It is quite painful to work on the API tests together with the whole team. Igors recommendation for large projects is to create powerful API tests with additional libraries like Requests (Python) or REST Assured (Java), which well describe in a bit.Postman pricing. Postman has four pricing plans:Free version for small projects with up to three usersBasic version (from $12 per user/month) for fast API designing, development, and collaborative team testing;Professional version (from $29 monthly) for the centralized management of the API workflow; andEnterprise version ($99 monthly) for API-first development.Licenses differ not only in the size of the feature list but also in the number of available monthly API calls to Postman API from 1,000 in the free subscription to 1 million in the Enterprise version.Requests: a human-friendly HTTP library in PythonRequests, an open-source Python HTTP library, was released under the Apache2 License with the aim of simplifying HTTP requests.Testing the response time to be less than the timeout via RequestsRequests advantages. The Requests library saves time and effort by fully automating keep-alive and HTTP connection pooling. No need to manually add query strings to URLs and form-encode POST data. Igor Pavlenko considers Requests a powerful library with easy-to-understand documentation, simple syntax, and rich functionality.It supports such features as OAuth 2 and Open ID Connect Authentication, various Certificates and much more. So, the choice is simple: If you have Python Requests is the best decision for you.REST Assured: a Java library supporting XML and JSON requestsREST Assured is a Java library for creating a REST API testing tool script. So, to get started with it, youll have to set up a new Java project first, and theninclude it as a library for your project. Its documentation is stored inside the GitHub repository. REST Assured is the best fit for functional testing of REST API services and it requires strong coding skills to create tests.REST Assured based test to check a GET API response, Source: Saurabh AgarwalREST Assured advantages. API testing is pretty hard in Java, and Rest Assured was designed to simplify things. With it, you dont have to create boilerplate code to test and validate complex REST responses. The library is open-source and has an active development community - another big pro.JMeter: a Java-based tool measuring SOAP/REST API performance and moreApache JMeter is a time-tested tool initially used for load testing. These days, Apache JMeter also supports functional, regression, and stress tests on different protocols. Its a free open-source tool with custom scripting functionalities that require advanced programming skills.Using JMeter for testing, Source: Saurabh AgarwalJMeter advantages. JMeter's pluggable timers allow for simulating various types of loads. Its plugins add extensibility and customization, while built-in functions enable dynamic inputs to a test. Thanks to its efficient UI design and lightweight component support, JMeter executes tests fast and provides accurate timing. It caches test results and data providing their offline replay and analysis.Oleksandr Fomenko, a Senior Test Automation Engineer at EPAM Systems, chose JMeter to measure the API performance for its convenient way of multi-threading emulation. You can create test cases using JMeters record and playback technology, extract and reuse data from response, verify it using JSON Assertion, debug it, and build a comprehensive report. With these functions in stock, JMeter can be easily used as a stand-alone framework for testing REST APIs.RestSharp: simple RESTful API testing in C#Available as a NuGet package, RestSharp is a popular and convenient open-source HTTP client library for . NET. Its easy to install and add to your C# project. RestSharp parses both XML and JSON formats.API test using RestSharp: Check whether the HTTP GET call returns status code 200, Source: BasRestSharp advantages. RestSharps functionality allows for straightforward test creation, serialization, and deserialization. It can handle synchronous and asynchronous requests with a wide list of ready-made authenticators. Uploading files and forms in multiple parts, RestSharp cuts down on upload times.EPAMs Oleksandr Fomenko highlights its powerful features and considers RestSharp the best solution for processing HTTP requests. He also accents its clean implementation: RestSharp allows you to easily implement any type of HTTP client.SoapUI and ReadyAPI by SmartBear: end-to-end testing tools for REST, SOAP, GraphQL, and other web servicesSoapUI is an open-source Java-based API testing tool by SmartBear for functional, security, load, and interoperability testing of SOAP, REST, GraphQL, and JMS (Java Message Service) web services.Its commercial version - ReadyAPI (former SoapUI Pro) - covers more types of functionality tests (load, stress, and spike). It also supports the validation of event-driven microservices powered by Apache Kafka. Ready API comes pre-integrated with Git, Docker, Jenkins, TeamCity, and other popular DevOps tools. (We have already described another SmartBear product used for UI testing - TestComplete - in our article on automated testing tools)SoapUI and ReadyAPI advantages. Both open-source and pro tools feature an easy-to-use drag-and-drop interface enabling scriptless creation and running of even the most complex test scenarios. They cover the full API testing spectrum. Yet, ReadyAPI has more extensive functionality including reporting and analytics. Also, as we mentioned before, it comes with out-of-the-box plugins for popular CI/CD tools.ReadyAPI pricing plans. ReadyAPI offers three types of annual subscriptions called modules, each covering a different set of features:API Test Module starts at 755 (around $796) and includes end-to-end functional and security testing. It also gives access to a limited number of functional testing and virtualization instruments;API Performance Module starts at 6,010 (around $6,339). It you to validate the speed, scalability, and performance of APIs, servers, and network resources. It doesnt include automated security testing, endpoint scanning, and some other instruments available in the API Test Module;API Virtualization Module starts at 1,131 (around $1,192). It allows you to virtualize critical APIs to reduce dependencies and delays throughout the delivery cycle. At the same time, the module lacks many testing features.Each option has a 14-day free trial. You can also bundle all three modules and get the entire range of features at a lower price.Katalon Studio: an automation tool with an inbuilt API testing modeDesigned primarily for UI testing, Katalon is also an integrated environment meant to easily generate and execute API tests.Katalon advantages. Its edge over other automation tools like Selenium or Ranorexis that it doesnt need additional integrations to test APIs. Katalon has an inbuilt API testing mode with the following capabilities:code auto-completion, inspection, and snippeting;references and debugger;inbuilt response viewer with auto-formatting;search for access to artifacts; andtests importing from API testing and editing tools (e.g. Swagger, Postman, WSDL).With Katalon, you can test all types of REST, SOAP/1.1 and SOAP/1.2 requests and multiple data sources.Katalon dashboard for testing APIKatalon pricing. There are three pricing plans:Free - for learning and experimentation. It covers 2,000 monthly test results;Premium - for production-grade testing workloads (from $25 monthly); andEnterprise - for large companies with mission-critical testing workloads, producing over 30,000 test results a month. Contact Katalon Studio for more pricing information.If Katalon has piqued your curiosity, we have an article with an objective review of this software.API testing framework vs ready-made API testing toolChoosing among the tools we described, keep in mind that some of them are better for writing an automation framework (RestSharp, REST Assured, and Requests), while other tools (Postman, SoapUI, Katalon, and JMeter) can be utilized as they are.So, which path to take?Writing your own framework gives you more freedom as youre not limited to the capabilities of a tool and its existing plugins. You can customize it your way: Add any library suitable for the chosen coding platform, introduce complicated logic in your tests, and build a reporting standard convenient to you. You also get more flexibility in managing your test data. But you enjoy these perks only if you possess strong coding skills.Otherwise, youd better look towards a ready-made tool. Its user-friendly interface and low-threshold coding requirements will be a relief for you. The pitfall here is that they usually address general API testing issues. But the good thing is that their community can easily help you solve your more-or-less common problems.To sum up, a ready-made tool like Postman is good for starters. Developing your own automation framework is a more advanced level, but its also rewarding in terms of functionality you can add to your testing solution. Copyright AltexSoft 2025. All Rights Reserved. In this digital age, APIs have become the cornerstone of how data is shared and processed. But many users are often unaware of the fact that they are putting their trust in an API and not a person.This is why it's important to leverage API testing techniques to ensure that every aspect of your website or application works as expected. My first interaction with APIs was while working on a personal project, a VirtualAssistant. It required me to fetch real data from third party apps. As a beginner, the knowledge I acquired was mainly from tutorials I had watched which basically covered how to GET, POST and DELETE data. I didn't know that there were rules that I should follow while working with APIs. But as time progressed I came to understand the importance of APIs and the major role they play in the daily applications of a developer's job.Mastering the rules is an essential step in learning about APIs. Most applications today are made up of pieces of software, each of which needs to be tested in its own way. There are always new and exciting ways to test an application and we're not just talking about checking for bugs here. It's more about the app functionality.As technology continues to advance, the trends related to how to test these applications will also change. In this guide, we will be discussing the trends that apply to the testing of APIs, how they differ from other types of testing, tools we can leverage when it comes to testing APIs, and how you can stay on top of your game when it comes to API testing.First, let's get an understanding of what we mean by APIs. APIs are designed for developers to use. They are, in essence, a coding tool that allows your application to communicate with other applications. APIs allow you to integrate third-party applications into your work or use your own data and processes in the cloud.The functionality of any application is defined by the connection it has with the outside world. For this reason, connecting your application with APIs becomes very helpful. In today's development process, APIs have become an essential part in web and mobile applications. They enable communication of different components of these systems and help you access data and services.A very important thing to keep in mind is that when dealing with APIs, you need to be sure everything works properly and that you get the correct feedback before integrating them into your applications. That's why testing them is essential.For this reason, a vital part of testing any type of software requires evaluating whether or not there's a way the app can be broken. In order to ensure a certain level of continuity when using an API in multiple ways simultaneously, you need to conduct API testing.Now that you have an understanding of what an API and why testing it is important, let's clarify what exactly might we mean by this is there more to the term testing? API Testing is a process that checks that an API adheres to the given requirements. Achieving this goal is not an easy task and there are many ways in which a tester can go about doing it effectively. You can conduct API testing manually or automatically, and it is often considered a part of integration testing. Manual API testing involves directly interacting with the API using tools like Postman, API Tester or any other available online tools. Automated API testing uses specialized software to send requests to the API and then compares the results to the expected behavior.API testing is important because it helps ensure that the various components of a system are working together as expected. By verifying the functionality of the API, you can be confident that the system as a whole will function as intended. You can use API testing tools to test the functionality of an API. They can help you test the performance of an API, as well as to check for security vulnerabilities. There are a number of different API testing tools available, each with its own advantages and disadvantages. Some of the most popular API testing tools include SoapUI, Postman, Runscope, and API Tester. Having a set of standard rules is the best way to ensure the quality of your APIs and their implementations. You can apply these rules during testing, coding, and development, as well as in production. Here are some principles that you should keep in mind:API testing should be a part of your continuous integration and delivery pipeline.API tests should be easy to maintain and write.A well-designed API will make your tests easier to write.You should test at the boundary of your system.Keep your tests small and focused.Make sure your tests are deterministic.Run your tests in parallel for speed.Use the available and freely accessible tools to simplify API testing. There are many different types of tests you can perform on your APIs:Functionality tests focus on ensuring that the API is able to perform its intended functions. Performance tests measure the response times of the API and check for bottlenecks. Security tests assess the vulnerability of the API to attack vectors such as SQL injection and cross-site scripting (XSS). To get started with API testing, you will need to have access to an application with an exposed API. You will also need to choose a method for sending requests to the API (manual or automated), and select a tool or framework for writing your tests (if using automated testing).Once you have these things set up, you can begin writing your test cases and running them against the API. API testing can be a challenge, but regardless of the tools you decide to use here are some tips that can help:Make sure you have a clear understanding of the API before you start testing. Read the documentation and any other available materials. This will help you know what to expect and how the API should work. Use API testing tools: This will give you a better understanding of how the API works and make it easier to find any issues. Test all aspects of the API, including input validation, error handling, and security. These are all important factors in ensuring that your API is working correctly.Keep your tests up-to-date as the API changes. This will help ensure that you catch any new issues that may arise. Use mobile apps for API testing. Things are changing all the time these days, where many people are working remotely, and from their phones too. By using mobile apps you can increase your productivity, become more mobile, and work from anywhere in the world. A good example of such a tool is the API Tester. By following these tips, you can help make sure that your API testing is effective and efficient. API testing is a key part of ensuring the quality of your software. By following these highlighted rules you will be able to ensure your tests are functional, secure and have a reliable performance. This will in turn help you improve the overall quality of your software applications and provide a better user experience. API testing is a critical aspect of software testing as APIs serve as the communication channels between different software components, allowing them to interact and exchange data. API testing not only involves validating the functionality, but also the performance, security, and reliability of APIs to ensure they meet the intended requirements and perform as expected. Ensuring that youve got complete coverage can be a challenge and that is why we have prepared this comprehensive API Testing Checklist based on our experience in delivering software testing services to our clients. Before we head to the checklist, lets understand the criticality of API testing and the prerequisites youll need to follow the checks as well. What Makes API Testing Crucial? Although we gave you a brief overview of the API testings importance in the introduction, it would be better if you understand it in detail to ensure you can modify our API Testing checklist as per your varying requirements. Functionality Validation: API testing ensures that APIs function correctly and perform the intended operations. It verifies that the API endpoints return the expected responses, handle different scenarios, and adhere to the defined specifications and requirements. Integration Testing: APIs serve as the interfaces between different software components. API testing helps validate the integration of these components, ensuring smooth communication and data exchange between them. It helps identify any issues or inconsistencies in the integration process. Performance and Scalability: APIs often handle a significant volume of requests and need to perform efficiently and scale seamlessly. So youll have to assess the APIs response time, addition, and resource utilization under different payload conditions. It helps identify bottlenecks, optimize performance, and ensure scalability. Security and Reliability: APIs are also potential entry points for security vulnerabilities and attacks. That is why it is critical to maintain their security by identifying vulnerabilities like injection attacks, cross-site scripting (XSS), and authentication/authorization flaws. It helps ensure that APIs are secure, protect sensitive data, and follow industry best practices. Version Compatibility: APIs evolve in time with new versions introducing changes and improvements. So it is important to validate the compatibility between different API versions and ensures backward compatibility. It ensures that existing integrations and applications continue to function correctly when API versions are updated. Error Handling and Exception Management: APIs should handle errors and exceptions gracefully, returning meaningful error messages and appropriate status codes. API testing verifies that error handling mechanisms are in place and that the API responds appropriately to different error scenarios. Pre-requisites for API Testing Even with the API testing checklist in hand, you will not be able to perform the testing directly as there are a few prerequisites that have to be done from your end. So lets see what those prerequisites are, Understanding API Documentation: Familiarize yourself with the API documentation available such as Swagger to understand the details about endpoints, parameters, expected responses, etc. This will play a crucial role in making changes to our API Testing checklist to align with your needs. Setting Up the Test Environment: Next up, youll need to ensure you have the test environment to do the tests. If a test environment isnt available, make sure to set it up or reach out to the concerned team to get it done. Identifying Test Data: The next part is having test data to cover valid and invalid scenarios, edge cases, and boundary values. Establish a systematic approach for efficient test data management, encompassing the storage and organization of test data sets for reuse and maintenance. Test Automation: Test the APIs manually to conduct exploratory and feature testing. But to speed up the process, you can focus on implementing test automation to execute the repetitive tests and save time. You can use tools such as Postman, Rest Assured, or other tools mentioned below based on your preference. Tools for API testing: Since both manual and automation testing is required, choose the right API testing tools for your needs. Heres a list of tools commonly used for API testing in both: Manual API Testing Tools Postman Swagger UI cURL Insomnia SoapUI Automation Testing Tools Postman (Automation) RestAssured in Java Requests In Python Karate DSL Fiddler By addressing these prerequisites, you lay a foundation for a well-prepared environment with the right resources to execute the API testing checklist effectively. Key Definitions If youve already worked with APIs, youll be familiar with these terms. But if youre just getting started, it is important that you are aware of these definitions to understand the checklist with ease. Endpoints: It is a specified location within an API that accepts requests and returns responses. Payload: The term payload denotes the information or data transmitted by the client in a request to the server, or the information provided by the server in response to a request. Request: It is a question or a demand made by the user to a computer, asking for specific information or action. Response: The answer or action taken by the receiving computer in response to the request. Query parameters: They are provided at the end of the URL and are used to filter, and sort the data given by the API. Key-value pairs: In key-value pairs, youll find a colon separating them, for example, key: value and the key remains static, serving as a consistent identifier. API Testing Checklist Now that we have seen the fundamentals, lets head directly to the API testing checklist. We have categorized the checklist to help you understand and perform these checks with ease. API Version We start our API testing checklist with API version validation and it is the process of ensuring that an API behaves appropriately and consistently across different versions. APIs are frequently updated, with new versions being published to add features, repair issues, or enhance performance. However, these upgrades can occasionally introduce changes that alter the APIs behavior. In API version validation, testers typically perform the following tasks: Testing forward compatibility: Check if older clients can still work with newer versions of the API. This ensures that new features added in the newer version do not break existing clients. Regression testing: Re-run existing test cases against the new version of the API to ensure that the core functionality remains intact and that new changes have not introduced any regressions. Response Status code The status code is an essential part of API responses as it indicates the success or failure of a request. Verifying the expected status code ensures that the API is functioning correctly and returning the appropriate status codes for different scenarios. Example: If we expect a successful response, we will verify that the API returns a status code of 200 (Success). On the other hand, if we expect an error response, we would check for status codes like 400 (Bad Request) or 500 (Internal Server Error). Lets take a deeper look at these responses in our API testing checklist now. 2xx Success Responses: These codes confirm that the clients request was successfully received. 200 OK: Signifying a successful request, the server returns the requested data. 201 Created: The server successfully processed the request, resulting in the creation of a new resource. 204 No Content: Although the request succeeded, the server does not provide any data in response. 4xx Client Error Responses: These codes signify issues with the clients request, such as mistyped URLs or invalid credentials. Prominent examples are: 400 Bad Request: The request is incorrect or invalid. 401 Unauthorized: The client lacks authorization to access the requested resource. 403 Forbidden: Although authenticated, the client lacks authorization to access the requested resource. 404 Not Found: The requested resource is not present on the server. 5xx Server Error Responses: These codes reveal that the server encountered an error while attempting to fulfill the clients request. Examples include: 500 Internal Server Error: A generic code indicating an unexpected condition preventing the server from fulfilling the request. 502 Bad Gateway error: It occurs when a gateway or proxy server receives an incorrect answer from an upstream server. 503 Service Unavailable: Issued when the server is temporarily unable to handle the request, often during high-traffic periods or maintenance. Presence of JSON Elements Next point in our API testing checklist is about JSON elements as API responses often include JSON data, which consists of key-value pairs. It is important to ensure that all the required JSON elements, or keys, are present in the response. This helps validate the responses completeness and ensures that the expected data is returned. Example: Suppose we expect an API response to include the following JSON elements: name, age, and email. We would verify that these elements are present in the response and contain the expected values. Data Types for Response Values API responses can contain data of different types, such as strings, numbers, booleans, or arrays. Validating the data types for response values ensures that the API returns the expected data types, which helps in maintaining data integrity and consistency. Example: If we expect a response value to be a number, we will verify that the API returns a numeric value and not a string or any other data type. Value Formats Similar to the data type we saw previously in our API testing checklist, some API responses may include specific value formats, such as dates in the format MM/DD/YYYY. Validating value formats ensures that the API returns data in the expected format, which is important for compatibility and consistency with other systems or processes. Example: If we expect a date value in the format MM/DD/YYYY, we have to verify that the API response follows this format and does not return dates in any other format such as DD/MM/YYYY or DD/MM/YY, etc. Invalid Request Headers When testing an API, it is important to verify how it handles invalid requests. Lets start this part of our API testing checklist with invalid request headers by checking whether the API returns appropriate error messages when invalid or incorrect headers are provided. Example: Suppose the API expects a valid access token in the Authorization header like this: Authorization: Bearer Now, during testing, you might intentionally introduce an invalid header, such as: Authorization: Bearer Testing with this invalid header helps ensure that the API responds appropriately to unauthorized requests. The API should return a specific HTTP status code (e.g., 401 Unauthorized) and provide a clear error message, indicating that the provided access token is invalid or missing. Invalid Request Body Now that we have seen how invalid request headers should be managed, lets check how invalid request bodies should be handled in our API testing checklist. When you send a request to an API, the request body often contains data in a specific format (e.g., JSON or XML). If the data in the request body is not well-formed or does not contain the mandatory fields, the API should respond with an appropriate error message. Example: Consider an API that expects a JSON request body for creating a new user. The expected format might look like this: { "username": "john_doe", "email": "", "password": "secure password"} Now, during testing, you intentionally send a request with an invalid key: { "username": "john_doe", "email": "", "invalid_field": "securepassword"} In this example, the invalid_field is not expected in the APIs request body. The API should detect this issue and respond with an appropriate error message. Header Parameter Limit APIs often have certain limits or restrictions on header parameters, such as maximum character limits. To ensure that the API handles such scenarios correctly, we can test by hitting the API with more than the expected limit for a header parameter and verify the response. Example: Suppose you have an API that expects a Content-Length header indicating the size of the request payload. The API may have a specified limit on the size of the payload it can accept, and exceeding this limit could lead to issues or security vulnerabilities. The expected header might look like this: Now, during testing, you intentionally send a request with a Content-Length header exceeding the expected limit: In this case, you are testing the APIs ability to handle oversized headers. The API should detect that the request header exceeds the defined limit and respond appropriately. Invalid Header Parameter Similar to sending header parameters beyond the defined limited, we have also included a check in our API testing checklist to see how an API handles invalid header parameters. It is important for maintaining security and data integrity. By sending invalid header parameters, we can ensure that the API rejects or handles them appropriately. Example: If an API expects a header parameter called X-API-Key, we can test by sending an invalid or non-existent header parameter, such as X-Invalid-Header: value, and check if the API returns an error or handles it correctly. Invalid Authorization Header Value Authorization headers are often used to authenticate and authorize API requests. Testing with invalid authorization header values helps in verifying that the API rejects unauthorized requests and returns appropriate error messages. Example: If an API expects an authorization header with a valid token, we can test by sending an invalid or expired token and check if the API returns an error indicating invalid authorization. Valid Content-type values in the Request Header Verifying an API request with valid Content-Type values in the request header involves testing how the API correctly processes different content types. The Content-Type header informs the server about the media type of the resource being sent or requested. Example: Suppose you have an API endpoint for creating a new resource, and it accepts data in JSON or XML format. The valid Content-Type values might include: JSON Content-Type: POST /api/resourcesHeaders:Content-Type: application/jsonRequest payload:{ "name": "New Resource", "description": "A description of the new resource"} XML Content-Type: POST /api/resourcesHeaders:Content-Type: application/xmlRequest payload: New Resource A description of the new resource Without Authorization Header Parameter Similar to checking invalid header parameters previously in our API testing checklist, it is important to test how an API handles requests without the required authorization header parameter. This helps ensure that the API enforces proper authentication and authorization. Example: If an API requires an authorization header parameter, we can test by sending a request without the authorization header and check if the API returns an error indicating the missing authorization. Expired Authorization Token When dealing with authorization tokens, it is important to test how the API handles expired tokens. By sending an expired token and hitting the API endpoint, we can verify that the API rejects the request and returns an appropriate error message. Example: Consider an API that requires an Authorization header with a valid and non-expired access token for authentication. A valid authorization header might look like this. Authorization: Bearer valid_access_token Now, during testing, you intentionally send a request with an expired access token: Authorization: Bearer expired_access_token In this example, the API should detect the expired authorization token and respond with an appropriate error message. The expected behavior might include an HTTP status code, such as 401 Unauthorized, and a response body with a clear error message Pagination As pagination is a common technique used in APIs to retrieve data in chunks or pages, we have included a check for them in our API testing checklist. When testing pagination, it is important to verify whether the API returns the expected amount of data based on the specified data count limit for pagination. Example: Suppose we want to retrieve 10 items per page using pagination. We would hit the API with the appropriate parameters and verify that the response contains exactly 10 items. Valid Query Path Parameters When verifying the response for an API endpoint with all the valid query path parameters, you are essentially checking how the API processes and responds to correctly formatted query parameters. We will also check for invalid query path parameters next in our API testing checklist. Lets consider an example now: Example: Suppose you have an API endpoint for retrieving information about a user, and it accepts several query parameters: Query parameters: userId (required): The ID of the user. includeDetails (optional): A boolean parameter to include additional details. A valid API request with all the valid query path parameters might look like this: GET /api/users?userId=123&includeDetails=true In this example: userId is a required parameter, and it is set to 123. includeDetails is an optional parameter, and it is set to true. The expected response from the API should include the relevant information based on the provided parameters i.e. userId: 123. { "userId": 123, "username": "john_doe", "email": "", "details": { // Additional details based on the includeDetails parameter "age": 30, "location": "City" }} Invalid Query Path Parameter Testing with invalid query path parameters helps in ensuring that the API handles such scenarios correctly and returns meaningful error messages. Example: If an API endpoint expects a query path parameter called id, we can test by providing an invalid or non-existent value for this parameter and check if the API returns an error indicating the invalid parameter. Special Characters in Query Path Parameter The next check with regard to query path parameters in our API testing checklist is with special characters as it can sometimes cause issues or unexpected behavior in APIs. By testing with special characters in query path parameters, we can ensure that the API handles them correctly and returns the expected response. Example: If an API expects a query path parameter called name, we can test by providing a value with special characters, such as John&Doe, and check if the API handles it properly. Request Payload Request payloads often contain data that is required for the API to process the request correctly. By verifying that all the required fields are present in the request payload, we can ensure that the API receives the necessary data. Example: Suppose an API requires a request payload with fields like name, email, and password. We would verify that all these fields are present in the request payload before sending the API request. Without a Request Payload Similar to other checks in our API testing checklist, we should also test an API request without a request payload involves testing how the API handles scenarios where no data is provided in the request body. Example: Suppose you have an API endpoint for creating a new user, and it requires certain fields in the request payload. But you didnt provide any request body, the API should handle this scenario gracefully and respond appropriately. The expected response might include an HTTP status code, such as 400 Bad Request, and a response body with an error message indicating that the request payload is missing or malformed. Without a Required Field in the Request Payload To ensure data integrity and completeness, APIs often require certain fields in the request payload. By testing without a required field in the request payload, we can verify that the API returns the expected error message or response. Example: If an API requires a request payload with a field called email, we can test by sending a request without the email field and check if the API returns an error indicating the missing field. Invalid Data Types in the Request Payload Next up in the set of request payload checks in our API testing checklist is to test with invalid data types in the request payload. APIs often have specific data type requirements for request payloads and so we have to ensure that the API handles them correctly and returns meaningful error messages even with invalid inputs. Example: If an API expects a numeric field in the request payload, we can test by sending a string value instead and check if the API returns an error indicating the invalid data type. Request Payload Length Similar to other limitations seen in our API testing checklist, APIs also have limitations on the number of characters or the maximum length allowed for certain fields in the request payload. By testing with values exceeding these limits, we can ensure that the API handles them correctly and returns the expected response. Example: If an API expects a field called description with a maximum limit of 100 characters, we can test by sending a value with more than 100 characters and check if the API returns an error indicating the exceeded limit. Null Value in the Request Payload Some APIs may allow certain fields to have null values in the request payload. By testing with null values for these fields, we can ensure that the API handles them correctly and returns the expected response. Example: If an API expects a field called address in the request payload, we can test by sending a null value for this field and check if the API handles it properly. Special Character in the Request Payload Special characters can sometimes cause issues or unexpected behavior in APIs. By testing with special characters in fields of the request payload, we can ensure that the API handles them correctly and returns the expected response. Example: If an API expects a field called Contact in the request payload, we can test by sending a value with special characters, such as 998877665$, and check if the API handles it properly. Valid Key-value Pair in the Query String Parameter Next in our API testing checklist, were going to see a sequence of checks with the Query string parameters that are used to provide additional information to the API endpoint. By testing with valid key-value pairs in the query string parameters, we can ensure that the API correctly processes and returns the expected response based on the provided parameters. Example: Suppose we have an API endpoint that expects query string parameters like category and sort. We can test by providing valid values for these parameters, such as category=books and sort=price, and verify that the API returns the appropriate response. Invalid Key-value Pair in the Query String Parameter Testing with invalid key-value pairs in the query string parameters helps ensure that the API handles such scenarios correctly and returns meaningful error messages. Example: If an API endpoint expects a query string parameter called page, we can test by providing an invalid or non-existent key-value pair, such as invalidKey=value, and check if the API returns an error indicating the invalid parameter. Different Data Types in the Query String Parameter APIs may have specific data type requirements for query string parameters. By testing with different data types in the query string parameters, we can ensure that the API handles them correctly and returns meaningful error messages. Example: If an API expects a query string parameter called count with a numeric data type, we can test by providing values of different data types as shown below, "GET /api/products?count=10" (valid) "GET /api/products?count=somestring" (invalid), It should return the appropriate error code or message when it is an invalid parameter. Valid Date Format Key-value pair in the Query String Parameter The final check with the query string parameters in our API testing checklist is with the valid date format. Some APIs may require specific date formats in the query string parameters. By testing with valid date formats, we can ensure that the API correctly processes and returns the expected response based on the provided date. Example: If an API expects a query string parameter called date in the format YYYY-MM-DD, we can test by providing a value like "GET /api/products?date=2024-02-16" We can ensure it returns the appropriate response message or code. Server Request Per Second Configuration Were now moving towards the performance part of our API testing checklist. To test the performance and rate-limiting capabilities of an API, we can hit the API multiple times within a short period to exceed the configured request per second limit. This helps verify that the API enforces the rate limit and returns the expected response or error message. Example: If an API has a rate limit of