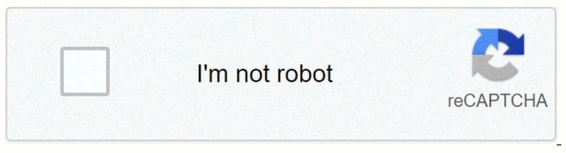


I'm not robot!

Api testing strategy pdf

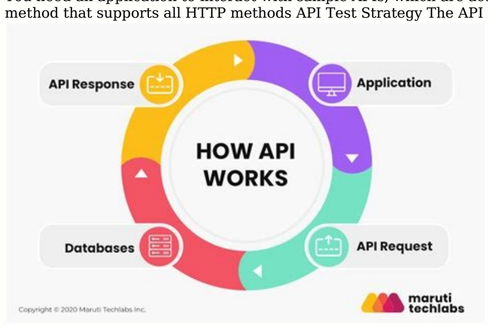
Api testing standards.

Api testing methods. Api strategy example. What is api testing for beginners.

Recent studies on customer experience by Zendesk reveal that one bad customer service experience leads to 39% of customers ignoring a company for the next two years. To avoid such incidents, you should shape up your user interface (UI) interactions effectively to engage with the prospects and constantly add value to their interactions with the content on your website.

It’s one of the fastest, most convenient ways to prevent massive losses and boost your profits. Creating an excellent UI is not possible without an Application Program Interface (API) - the intermediary that bridges gaps between programs. One type of API called Representational State Transfer (REST) API can transform businesses of any size or industry by addressing the needs of both companies and consumers. Read on to learn how to use REST API to achieve new heights. What Is an API? API is a set of rules for software that validates the capabilities of a new application in terms of performance, operations, and security. It uses rules to enable programs to connect to other files. Developers create APIs on servers to activate communications with clients. This software intermediary is present in many of our daily tasks. For example, you can witness how API works in sites that offer several login options. If a page shows you options to log in via Facebook, Instagram, or Twitter, it uses an API to provide applications with identification information. In this scenario, the API elevated the customer experience by making the login process convenient and seamless. What Is REST? Developers created REST - a software architectural technique for designing networked applications - to guide the development of the World Wide Web. This method focuses on reducing latency and boosting security. Today, many software companies use it as a set of guidelines for creating reliable web services. REST developers use standard operations with a stateless protocol. This process creates fast, reliable services that allow specialists to reuse REST components. You can update and manage these factors even while a system runs. What Is a REST API? A REST API is a type of API that adheres to REST limitations and allows interactions with other sites. Any page that follows REST constraints is informally called RESTful API. Below are some of its core features. A RESTful API is stateless It supports JSON and XML alike It offers simplified implementation versus the standard SOAP Every architectural modification in the REST API should reflect in its documentation It provides developers with an extensive range of error messages Here’s a REST API example in an e-commerce setting. If you want to purchase affordable running shoes, you use a search engine, type in keywords, and receive a list of options in return. In most cases, the results are relevant to your search. If used correctly, this tool should enhance a client’s experience with your brand. Before you run tests, here are some guidelines to follow on how to create REST API: Use HTTPS Include a timestamp to requests Limit HTTP methods Apply input validation if necessary Use OAuth Avoid showing sensitive data on URLs Create a security checklist and run tests How To Test REST API The goal of testing REST API is to check individual functions.

You need an application to interact with sample APIs, which are activities that require a testing tool and a code. You can use these tools to test REST API cases: Postman: A scalable instrument that integrates into a CI/CD pipeline Curl in Linux: A command-line mechanism that supports over 20 protocols in transferring data Advanced Rest Client: A method that supports all HTTP methods API Test Strategy The API test strategy provides information on individual test scenarios and cases. Below are three core objectives of API functional testing. APIs Ensure Proper Implementation: In software development, complex codes may sometimes lead to bugs.



It’s impossible to prevent bugs when building software, but an API can help check a system’s functionalities to detect and fix them. They Ensure that Executions Work as Intended: This API function later becomes your documentation. It provides updates on an API’s lifecycle, including new versions, upgrades, and limitations. They Provide Regressions Between Code Mergers and Releases: The objective of any file is to merge its algorithms with other programs without conflict. However, changes may sometimes cause friction. API enables developers to enhance features seamlessly. What to Test in an API APIs are essential in gathering data and implementing processes for customer service improvements. It's a vast, complex field, making it crucial to know what you want to test. This step is essential to avoid getting overwhelmed with all the available information. Below are some of the most common factors to consider. Several test actions make up one API test, making them essential in a test flow. Every API request should include the following: Verify HTTP Status Code: This step is crucial to quickly finding and fixing broken links. You can create a resource to return un-permitted requests like redirections, client errors, and server errors. Verify Response Headers: Response headers are HTTP requests that don’t relate to the message. You can use factors like age, location, or server to gain more context about a response. This step is crucial because it affects both the security and performance of your site. Verify Response Payloads: Apart from letting you see replies from curl or Postman, this action will help you make use of JSON data. The information should include field names, types, values, and errors. Verify Basic Performance Sanity: Sanity testing makes sure that there are no bugs in the system and that they will not affect code changes. It also checks for the completion speed of any test. Verify Application State: This action may be optional, but we highly recommend taking it. You can use it for manual testing or when you have access to an interface such as UI. API is a fundamental aspect of any software application, (like healthcare apps, for instance) making it prone to attacks. Many developers use this tool, which means it provides access to sensitive functions and data. It’s crucial to run authorization and security checks to enable safe transactions for your clients. Check that APIs Follow Proper Security Protocols: Here are four aspects to consider when you’re enhancing your site’s safety: parameter tampering, injection, input fuzzing, and unhandled HTTP systems.
Oversee Role Permissions: You can restrict user access depending on an individual’s job description. Evaluate the responsibilities team members have to determine the resources they need. Don’t permit anybody unnecessary any access that might compromise your program’s security. Find Data Leaks: Data leakage is the unauthorized information transfer from an internal source to an external destination.

Whether it’s intentional or not, it can compromise your reputation, finances, and operations. Because of its functions, it’s essential for APIs to be accurate, secure, and reliable. There are many types of API tests, from UI to security and validation testing. In most cases, these tests fall under the performance category because they aim to determine the effectiveness of certain aspects of the site. They help organizations collect performance data to improve customer experience. Software developers perform load tests to evaluate system performance using real-life load conditions. On the other hand, specialists run stress tests to verify a system’s robustness under extreme conditions. While the two test types have distinct differences, they both have a common goal. They aim to determine the breaking points of sites and applications to avoid crashes. Usability tests, also called user experience tests, measure the user-friendliness of a software. Experts run them to determine and fix defects that hinder seamless consumer operations.
Remember to test the entire customer journey when using them, from logging in to authentication and purchase. Any error in the process can lead to lost clients. Conclusion In today’s modern era, organizations have to provide their clients with secure and scalable connections between a diverse range of platforms. REST API testing is the only reliable way to achieve this goal. From protecting users from malicious code to maximizing time efficiency, this tool can transform your UI. As a trusted quality assurance provider for almost 20 years, QASource has grown by as much as 50% annually. We achieve unmatched growth levels because of our groundbreaking services. If you’re ready to take the digital world by storm, request a free quote now.
We can’t wait to help you take your business to new heights. Mohamed Fahmy 7 April 2022With the rise of connected applications, connected devices and cloud computing, the number of APIs has risen exponentially.Exposed and open APIs have been developed by cloud service providers (CSPs), service platforms, commercial off-the-shelf products (COTS) vendors, device manufacturers and others to help build digital ecosystems and make it easier for customers to consume their services.An application programming interface (API) is a connection between systems, applications, devices and infrastructure (typically cloud based). It is a type of software interface that offers a service or a capability to other software. Because of this connected nature, testing APIs differs slightly from testing traditional pieces of software.In this blog, we’ll detail various elements that should be considered when testing APIs—to provide higher quality APIs released to production. This should help speed up integration projects within enterprises and allow faster consumption of digital service providers.What Is an API First Approach?More and more organisations are now adopting an API first approach by creating open, robust and discoverable APIs.An API first approach means that when systems are built, capabilities, features and services are always made available by APIs so they are exposed to other systems and/or organisations rather than being made available by a graphical user interface (GUI) or other mechanisms. This helps organisations integrate their internal IT stacks providing value to their end customers. It also allows multiple organisations to create a value fabric of a digital economy offering add-on services, market places and other digital services.4 Types of APIAPIs now come in many different forms, with the following four types exploding in popularity:1. Application APIsApplications and system APIs that expose the capabilities and services of those systems to other internal/external systems, other organisations such as customers, partners and vendors or even third party independent developers.2. Cloud Services APIsCSPs offer their customers cloud services APIs to be able to consume their compute, storage, network and other cloud services.3. Hardware APIsThese are APIs exposed by original equipment manufacturers (OEMs) and devices manufacturers exposing features and capabilities to software developers and software vendors.4. Operating Systems APIsThese APIs refer to those exposed by operating systems like Android and iOS to allow the development of native applications to the operating system for a better experience and performance.Why Test APIs?The API landscape has seen a big shift with the surge in popularity of the above API types. Plus, APIs that already existed have dramatically changed from initially being proprietary and closed to now being open and discoverable.Enterprises are creating more and more APIs to integrate their own IT applications within their stack, to expose their own services they sell or to create an ecosystem for other enterprises to sell their services.API testing and integration testing is therefore more important than ever to guarantee high quality software; that’s because the API is either the gateway to an enterprise's service that they sell or the glue that holds their internal systems together to support their business processes and customer journeys.API and integration testing are usually the most painful parts of any software project. We now have the systems deployed, but they don't integrate well! During this phase of any project, the most friction between systems, platforms and organisations is discovered. As with any software project, it is cheaper to discover bugs and issues at the earliest opportunity during the development process.In this article we will go through the API testing strategy that makes it easier for organisations building APIs or integrating them together to test and deliver high quality APIs.API Testing Vs Other Software TestingAPIs are gateways to the capabilities of a system, application, platform or equipment. They are connected by nature. That connection is twofold: internally with the backend—be that a microservice, a database, or a business process managed by that application—and externally with the third party calling that API. With this connected nature comes the complexity of testing APIs.However, when we look at software holistically, in an enterprise architecture, there aren't any standalone applications or systems. Most of those systems already expose and call APIs. This means that the complexity of APIs already exist in IT stacks. So let us apply the existing software quality best practices to APIs while being mindful of their connected nature—this should yield to higher quality APIs.Let’s examine the various test phases in a software development life cycle and highlight some API specific tests that should be considered such as contract testing, discoverability testing and integration testing.Diagram 1: An architecture diagram of connected applications and platforms10 Tests to Conduct Across the Full CI/CD Pipeline1. Unit TestsThe first step in ensuring high quality software is creating a test centric environment.
Adopting test-driven development (TDD) for developing APIs ensures that there is a clear acceptance criteria for each development ticket, that is testable and well understood by the team.
The developers can then start by developing the test cases that fulfil the acceptance criteria, running the test cases that will fail, and developing the API with the goal of passing the tests. Those can be expressed as API specifications or a contract.Unit tests can be used to validate the functionality of the APIs early on. They should be run in isolation from any backend and third party connection which means that each test focuses only on the API itself. It validates on the input and output parameters, their formats, and on which fields are mandatory vs optional.The unit tests should validate the order of responses if the API has more than one response and any other tests related to the transmission of the message. Unit tests should not focus on the business/service logic behind the API.Unit tests serve as the first feedback on the code’s basic functionality—a gateway to the service/capability being exposed. Unit tests should be run quickly at the build stage; ideally each test should run in milliseconds. This should provide the quickest feedback loop to the developers where it is critical to fix any defect found that breaks the build immediately to maintain the flow within the pipeline. The build should not pass the unit tests if it is below a 100% pass rate.2. Contract TestingAn effective way to perform the unit tests is contract testing. Contract testing aims to validate that what is written in the API specification is actually implemented.The contract testing approach is a three phase test. It tests each integration point for a given API/provider in isolation and the Integration parameters from the point of view of each client/consumer for that API/provider against a mock (a software that mimics the API/provider) of the API/provider.Phase 1First, a test is run for all the possible input parameter permutations and business scenarios for that API/provider, for each known client/consumer. At this point, the test assesses how the client/consumer responds to the expected responses received from a mock.
Then the results are stored in a database and referred to as a contract. This phase is passed when we are sure that all known clients are happy with the expected responses and do not break.Phase 2In the second phase, we switch to test the API/provider’s point of view by triggering mocked requests to the API/provider that covers the same possible parameter permutations and business scenarios coming from all known clients/consumers. Here the tests prove that the API/provider is able to handle all possible requests from all known clients/consumers.Phase 3The responses are then stored in the database and finally checked against the results from the first phase looking for matches. This third phase ensures that both the clients/consumers and API/provider are both behaving in the same way they are expected, as it is defined in the contract. There are some tools that can facilitate contract testing (more on that in the testing tools section below).Diagram 2: Contract testing example3. Backward Compatibility TestingAPIs will have more than one version. As the software they front evolves, and new capabilities are added, new fields are added and capabilities exposed. Sometimes, some are also removed.This is not limited to functional features but could be extended to non-functional such as security protocols and the API protocol like simple object access protocol (SOAP), representational state transfer (REST), etc. A client calling the API using the older version could easily break if the newer version is not backward compatible.This introduces a new set of tests that checks the backward compatibility of an API. This is basically running the older versions of tests we already have run in previous iterations. This of course increases the complexity of API testing as the tests must be version controlled, retained and kept for the future.However there should be a line at which backward compatibility stops, as the higher the number of previous versions supported, the higher the cost. A good starting point is to decide on a number of set previous versions (e.g. the last four versions) or a given duration (e.g. 30 months).Once those test phases are complete, the software/API should be promoted to the next higher level environment. Those tests could be part of the unit tests, where simply older unit tests are run. The team should validate the test cases to make sure that the right compatibility tests are run and avoid false negatives.4. Integration TestingOnce the build is complete and the software is deployed into the next higher environment, “Development”, the next automated integration tests can be run.Integration tests can test the capability of both the software service/capability itself and the API combined. This can be done by simulating a calling system—sending preset input messages and examining the responses. Doing so requires the set up of test data within the tested system as well as a mock system for the calling system.Integration testing is more expensive as it requires more resources, data preparation and setup as well as the mocks (there could be more than one). This means that the team must be selective on the scenarios/test cases to run.Behaviour driven development (BDD) could be used where the test cases are representing the most important scenarios or use cases of the API under test. Again this should be agreed within the team including all stakeholders.Diagram 3: Integration testing5. End to End TestingOnce the integration testing is complete with the accepted exit criteria, the API or the software should be promoted to the next higher environment. This second environment should contain all the applications or platforms that are being integrated to enable end to end testing. It is a more expensive environment than the previous lower environment, hence the team should be conscious of the running durations for cost reasons.Now, it is the first time we test our integration end to end without a mock. This means that the team needs to prepare the test data that will be used in the testing, making sure that all involved platforms are deployed, up and running and connected with all security and networking prerequisites checked.The end to end (E2E) testing should be automated and included in the automated test suite designated for this step of the CI/CD pipeline. Following the same BDD approach, the team must agree on the scope of the E2E testing. Running those tests will be time consuming due to the connected nature of the test.Ideally, you should also agree on the exit criteria for the tests and the level of accepted issues found. We are not advocating for releasing lower quality software but the team can be pragmatic about the accepted level of quality based on the nature and criticality of the application as well as the severity of the encountered issue. Once the automated E2E tests are successful and complete, we can still run more tests in the same test environment.Diagram 4: End to end testing6. Manual Exploratory TestingOnce all automated tests are complete, and we are sure that all the known paths, scenarios and areas of focus are covered, it is time to add another layer of quality to our software, by manually exploring.Typically automated tests will cover pre-thought scenarios either defined in the acceptance criteria, or by past experience of the team. Exploratory testing here allows the freedom and the innovation of testers to go and find new scenarios that were not thought of and possible edge cases (that might not be very edge).Testers will already have the skills required to conduct exploratory testing after years of testing to find bugs. However it is key to timebox this exercise so we don’t compromise on the project time and subsequent cost or other team deliverables. It is important to define the exit criteria for this test phase so the team does not get hung up on real edge scenarios that are unlikely to happen compromising on the project timelines.For exploratory testing of APIs, there can be two approaches:Approach 1: Using an API simulation tool with a GUI allowing the user to enter the input parameter (the payload under test) and fire a messageThis could be quite an effective way of exploratory testing. However, the tester must be wary of the business rules/logic to avoid generating false negatives. The tool will allow you to enter any type of data that could be valid syntax wise but wrong from a business perspective.For example, I have previously worked on a project where a tester was calling the ‘View Bill’ API of a mobile service for a telecom operator; however the test subscriber used was a prepaid subscriber that does not receive any bills! The API was rightly returning an error, however the tester raised it as a bug.Approach 2: Using the GUI of the application to trigger the scenario that sets off the APIThis will protect against violating business rules; however, this might not be always possible as not all scenarios are triggered by a GUI and not all applications have a GUI to start with, especially backend systems.7. Discoverability TestingDiscoverability is a non-functional requirement often attributed to APIs, especially in the rise of open APIs and the inevitability of systems integrations in any given project. In this digital age, an organisation wants developers to quickly use and call their APIs to drive revenue for platform providers and make the software of software vendors more usable by their customers. This is a far cry from the closed proprietary APIs from 10-15 years ago.We should therefore test how easy it is for a developer to discover and use an API. Is the documentation up to date? Discoverability testing is a form of user acceptance testing where the user in this instance is a developer instead of an end user.8. Non-functional Requirements TestingOnce the API reaches the final lower environment, or “Pre-Production”, more non-functional requirements can be tested.The pre-production environment should be production-like and it is typically where performance, load and high availability testing can be performed. Since the tests that we run before this stage are automated, those could be scheduled to run with much higher intensity to test the load of a platform i.e. 30,000 requests per second.However, we should also be conscious of the other elements under test here such as the backend system or database the API is fronting, and whether there is any latency within the environment—especially in cloud infrastructure. Are we load testing the API or the application or platform behind it? This could be very tricky to isolate when performance or load issues are faced.If a given API that is connected to a database had poor performance during the testing, does the database query the API is triggering need to be optimised or does the main table in that database needs to be indexed? There is no clear cut answer to those issues when they appear. The team needs to be aware of the holistic solution being testing and start their troubleshooting accordingly.9. Security TestingSecurity testing isn’t always highlighted when discussing testing strategies, but it should never be an afterthought. Code scanning should be applied at the early stage of the software supply chain to look for vulnerabilities, smelliness, complexity and other attributes.At this later stage, the API is deployed in a production-like environment. More realistic tests could be performed such as identity related tests, authorisation tests, rate limit tests, penetration tests, and negative scenarios to check for the existence of the relevant security traps, as well as sending logs in a timely way to the Security Information and Event Management (SEIM) solution and others.Once those tests are passed, the API can be safely deployed in production. This could be done automatically or manually based on the level of maturity of the organisation as well as the criticality of the project.Diagram 5: API testing CI/CD pipeline example10. Production Testing and MetricsOnce the API is in the production environment, it can still be further tested before being released to the entire user base. There are a number of release strategies that are well beyond the scope of this blog so we will just highlight some of the most suitable ones.Depending on the project and the nature of the application, a team can choose to release an API to a specific set of users i.e. friendly or internal developers for a set duration, before pushing it to the entire base. The release strategy could be geographically based which works well for global organisations. Another way is to randomly select a small set of users as a test group before rolling it out.Both of these strategies involve gathering feedback from a small set of users before deciding to release the API to the general population. This must be a time bound activity, with a clear way of gathering the feedback and acting upon it i.e. the team must be authorised to release it to the entire base, roll it back or make any needed changes based on the gathered feedback.This feedback should be continuous even after releasing to the API. Production metrics must highlight the performance of the API, as well as its usability and reusability. Is it bringing back the business value promised during the planning phase? This feedback should direct the APIs roadmap. Should we further enhance this API? Do we create more APIs related to this capability?4 Important Considerations for API Testing1. Tests Should Be Incorporated Without Disrupting the FlowAfter going through the various test types and phases tested, those tests should be incorporated into the existing CI/CD pipeline without any disruption to the software supply chain. Possibly, there will be dependencies on mocks, test data preparation, test version control and third party apps that have to be managed.However it is key to start planning those interdependencies through the pipeline; agree on the entry and exit criteria for each among the team; and agree on the level of automation, code coverage and approval mechanisms (manual or automatic) for each phase. This will ensure the flow of new APIs through the development pipeline without any disruptions.2. You Should Be Aware of Enterprise Service Bus TestingIn real life complex technology stacks that are typical of a retailer, telecom operator or a bank where there is a “system of systems”, most organisations opt for an enterprise service bus (ESB) architecture which could be in house developed or a COTS product.Arguing for or against an ESB is outside the scope of this blog. However, in the projects or architectures that include an ESB, the team should be aware of including it in their testing plans.One common pitfall is to leave that part to the ESB team (if it is a separate team)—creating a silo. The result is usually project snags during the end to end testing phase. This is usually due to API orchestration issues, receiving API responses out of order (or not receiving at all) and numerous issues that should be addressed during testing.Ideally the team creating the API should be working together with the team using it. This is well beyond testing and starts with the project roadmap, requirements and even development plans. However when it comes to testing, considering the ESB when planning the end to end testing is a good approach to avoid integration issues often seen in projects with bigger integration scale.Testing an ESB isn't any different from testing a single API. Each endpoint that exists within the ESB (client facing, end-system facing or internal) should pass the same test process explained above. This way each end point is thoroughly tested.3. You’ll Need More Than One Test Environment During Each PhaseAs we went through the different test phases, it is recommended to have more than one test environment—typically the developer’s machine for the unit tests, the development environment for the integration tests, the Quality Assurance environment for the end to end tests, manual exploration tests and backward compatibility tests and the pre-production environment to run the non-functional testing such as the performance, load and security tests.Each environment will need its own assets in terms of infrastructure: compute, storage and networking. It will also require a set of applications like the testing tools, the backend that the API serves, the customer (calling App), and the mocks that will be used to test the API.Not all assets will be needed in each environment, so the team must plan in advance the assets needed for each environment as well as the times and duration each environment will be needed.These environments could be costly if not utilised properly. So an investment in startup and shutdown scripts could save the budget in the longer term. Those could be automated to the working hours of the team as well as called on demand when needed. The team must be conscious of the test data when creating such scripts, so the data is not lost or corrupted on the restart of a test environment.Diagram 6: Other test assets needed for API testing4.
Behaviour Driven Development TestingAn exhaustive list of test cases is never feasible within the time and cost constraints of a given project.
Adopting a BDD approach is the most economic option for selecting which test cases to prioritise and build. BDD should be done with the rest of the team (testers, product managers etc.) and should usually be defined within the acceptance criteria.APIs usually offer a certain type of service provided by the application. This could be a business service

such as a product list API of an Ecommerce platform, a suspend subscriber API of a Telecommunications billing platform or a technical service such as create folder API of a cloud storage service, or a shutdown API for a cloud compute service.As 100% code coverage may not be achievable, BDD is a great tool to make sure that the most critical scenarios are covered by the tests—like a basket and checkout API of an E-Commerce system or one that services the larger base of customers/users.Finally, the team should consider prioritising the section of code that historically generated the most defects.
This section of the code must have the highest—if not full—coverage. This will be known to the team after a few iterations of the project.API Testing ToolsThere are several testing tools and API specific tools out there. Comparing these tools is beyond the scope of this blog but we will name a few tools that offer a great starting point for API testing.Contract TestingPact (open source) and Pactflow (commercial):Language agnostic and contract testing specific.Both have a proper roadmap of features which means that the capabilities will continue to grow.The open source version, Pact, is relatively limited in features and in team sizes. The commercial version, Pactflow, offers different price plans suitable for all businesses and sizes.Spring:Free open source provided by VMware.It is java specific development framework, more suitable for java applications and and if the team is after their other capabilities such as applications tracing, microservices management, API management, cloud configuration management and others.It is free to use and VMware offers enterprise grade support, formal training and certifications.Integration TestingHummingbird: Hummingbird provides a test simulation tool available via command line and a GUI. This enables integration testing automation as well as manual exploratory testing.It can integrate with the CI/CD server in use and be triggered through the pipeline.It has a nice graphical display of the test performance and tests. This could be exported in JSON format to be integrated into the test automation tool in use.It is an open source project without enterprise support. The last publication of the project was at the end of 2019 which raises questions on the continuity of this project.SoapUI:SoapUI is a commercial product that started as an open source project.It provides a functional testing tool for SOAP and REST protocols.It comes with an easy to use GUI that allows to easily and rapidly create and execute automated functional, regression, and load tests.It can easily build up the needed mocks for the integration testing.SoapUI offers a free open source version; SoapUI as well as commercial products ReadyAPI and TestEngine with different price plans for each module and special pricing for bundled modules.Wiremock (Open source) and Mocklab (Commercial):Both offer a mocking service capability that is used in testing APIs.It supports both automated and manual testing as well as negative scenarios such as delays and faults testing.The commercial product comes with more capabilities and enterprise support with different pricing plans.Postman:This is a complete API management framework that simplifies each of the API lifecycle from design, build to test and monitor the API.One of the main features is the API repository that easily stores and catalogues the API artefacts on one central platform.Postman can store and manage API specifications, documentation, workflow recipes, test cases and results and metrics.Postman can be used to write functional tests, integration tests, regression tests via the GUI for exploratory testing or via the command line. This allows the integration of tests with the CI/CD pipeline. It also supports the building of Mock servers needed for the integration testing.Postman offers a free version as well as different paid price plans suitable for various organisations.Load and Performance TestingSoapUI:SoapUI provides load testing capabilities as part of their commercial product ReadyUI.It has features like parallel testing, reusing the functional tests, flexible load generation and server monitoring to monitor the server under test.SummaryAs we've shown, testing APIs isn't that different from testing software. DevOps practices should always be employed when developing and testing APIs, and teams should also try to adopt TDD and BDD approaches throughout. The API should be tested all the way through the software supply chain with the aim of finding any defects early enough when it is cheaper to fix.Once each set of tests are successfully passed, the API can be promoted to the next environment for more automated tests until the API is deployed in the production environment.After production deployment, the team can adopt a release strategy that allows for early feedback with actionable insights to determine what to do next. The team must be empowered to make the decision of releasing, rolling back or adding minor changes to the API based on that feedback.After the API is generally available, production telemetry is invaluable. This will help the team understand the usability, reusability and actual performance of the API. That way the team can plan the API roadmap and enhance the API further, add new complementary APIs or even retire the API if it is not used.