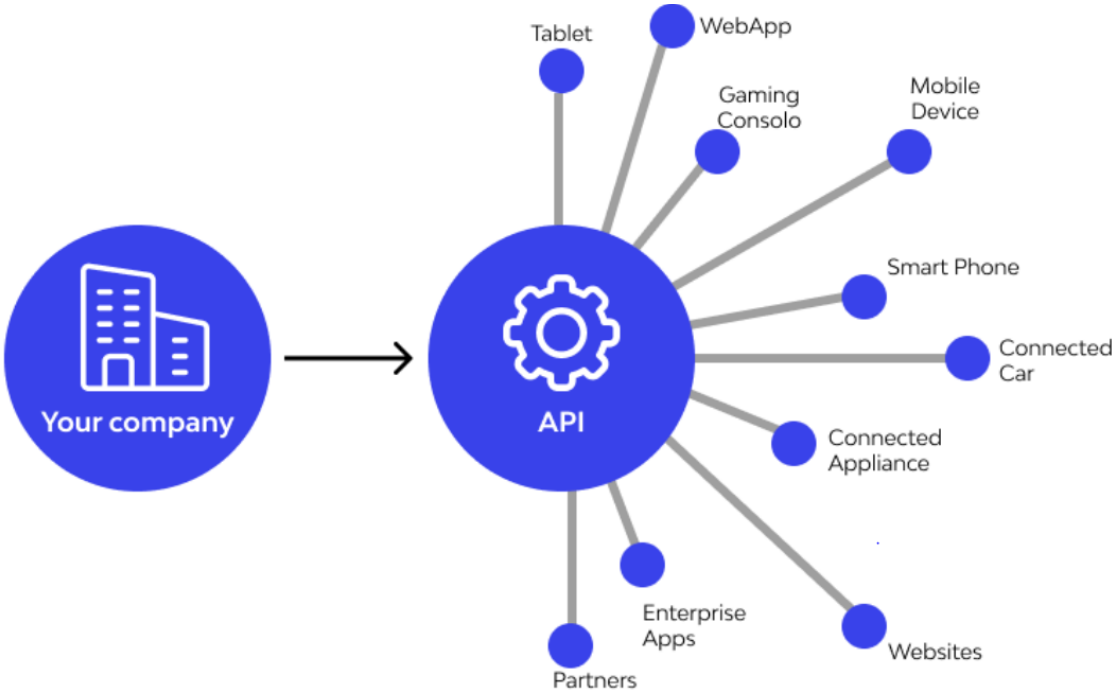
API Testing



Fake API Inputs and Outputs: https://jsonplaceholder.typicode.com/guide/

API testing Types?

Validation Testing

Validation testing occurs among the final steps and plays an essential role in the development process. It verifies the aspects of product, behavior, and efficiency. In other words, validation testing can be seen as an assurance of the correct development.

Functional testing

Includes testing particular functions in the codebase. These features are the representation of specific scenarios to make sure the API functions are handled well within the planned parameters.

UI testing

UI testing is defined as a test of the user interface for the API and other integral parts. UI testing focuses more on the interface which ties into the API rather than the API testing itself. Although UI testing is not a specific test of API in terms of codebase, this technique still provides an overview of the health, usability, and efficiency of the app's front and back ends.

Security testing

This practice ensures the API implementation is secure from external threats. Security testing also includes additional steps such as validation of encryption methodologies, and of the design of the API access control. It also includes user rights management and authorization validation.

Load testing

Load testing generally occurs after a specific unit or the whole codebase has been completed. This technique checks if the theoretical solutions work as planned. Load testing monitors the app's performance at both normal and peak conditions.

Runtime and error detection

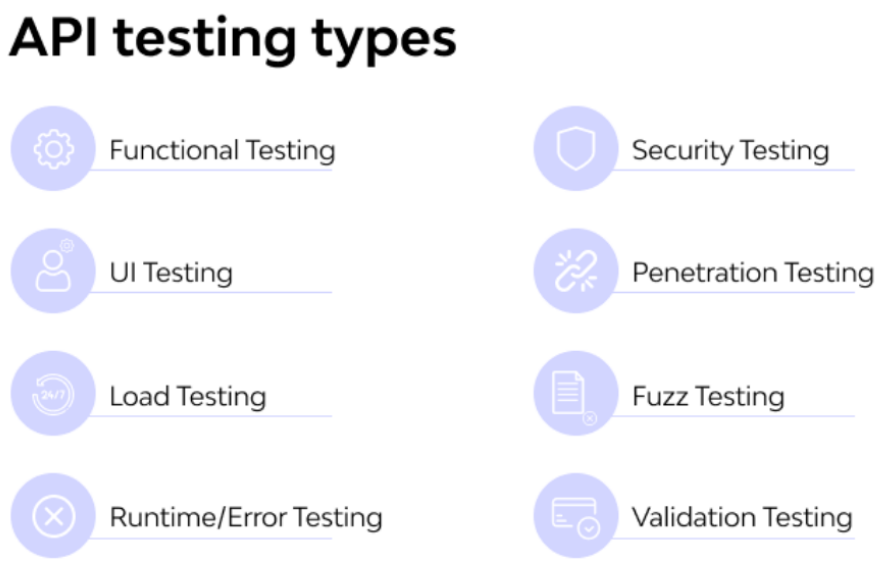
This testing type is related to the actual running of the API — particularly with the universal results of utilizing the API codebase. This technique focuses on one of the below aspects: monitoring, execution errors, resource leaks, or error detection.

Penetration testing

Penetration testing is considered the second test in the auditing process. In this type, users with limited API knowledge will try to assess the threat vector from an outside perspective, which is about functions, resources, processes, or aim to the entire API and its components.

Fuzz testing

Fuzz testing is another step in the security audit process. In fuzz testing, a vast amount of random data (referred to as "noise" or "fuzz") will be input into the system to detect any forced crashes or negative behaviors. This technique tests the API's limits to prepare for the "worst-case scenarios."



Why API Testing matter

Now you might be wondering why API is so important and that is understandable. After all it costs a lot of money and is not something you will always see the direct return of investment in. If you neglect to do this correctly however, you may find that the costs of the defects rack up much higher than the cost of the testing ever could.

Since APIs sit at such a central location, they also have most of the traffic to process and if they fail in unexpected ways, the consequences might be dire. A failure of any API could lead to services not being available, processes not working as expected and even allow access to objects and data that should not be accessible.

Another often overlooked aspect of APIs is the fact that it's not just the user facing functionality that needs to be tested, but often our APIs also integrate with other services or 3rd party providers. These all rely on our APIs passing expected and sane data.

Benefits of API testing

When we perform our API testing, one of the biggest benefits we can enjoy is that we can often test early. This is because the API component is often developed before the UI component which allows for swiffer feedback as well and it can help steer development on the components that integrate with our API or even adept our API if need be.

API testing is also a lot cheaper than manual e2e testing or automated UI testing because we can create much more fine grained component tests that we do not have to repeat every as often as our end to end scenarios.

All of these benefits allow us to leave enough room for security testing where it is most impactful, the API level. Security testing on all levels is useful of course but we do not live in a world with endless budgets so we have to put our priorities in order.

If we have tested our APIs well and documented them properly, we can allow for a much smoother integration and we can ensure that whatever interface subscribes to our API, they are well informed and ensured of a tested API.

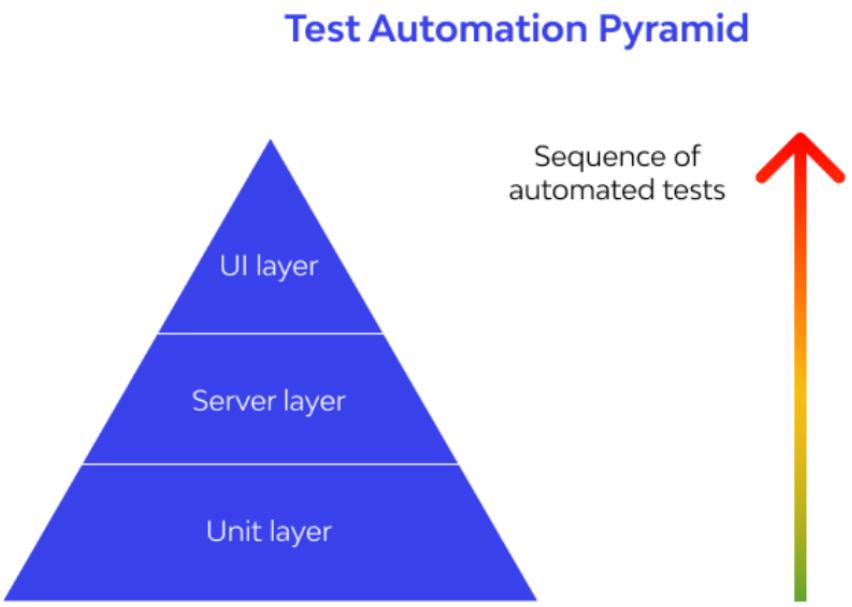
How to get started with API testing

1. Define the API requirements: Writing good requirements is very hard. Open communication is important to define proper requirements as every stakeholder can have useful input on the requirements and some very tough questions need to be answered to which no single human can have the answer. This is why teamwork matters, developers, testers, system architects, analysts, product owners... they all know their domain best and should be included in the creation of requirements.

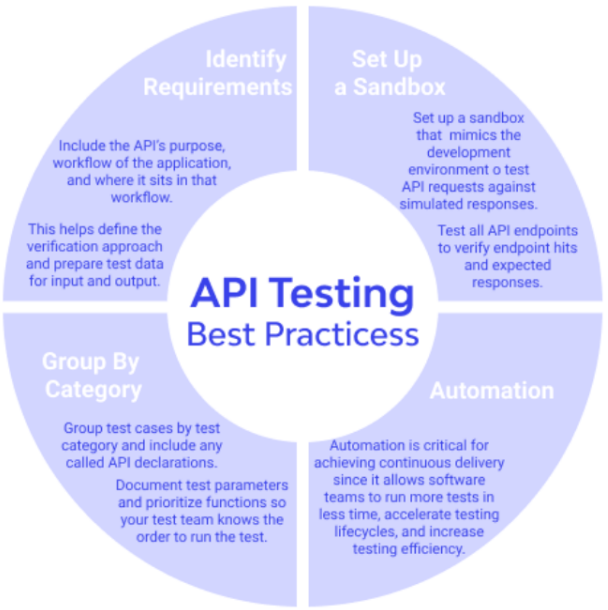
- What will this API be used for?
- How will the application handle data?
- How will the system handle failure?
- How will the system handle output?
- How will the system handle unexpected input?
- What are the requirements for the fields the API sends out and receives?
- How the API will interact with other APIs, such as which protocol will be used,...
- Entry and exit criteria
- The pass and fail criteria for the API/feature

2. Next we need to set up a good testing environment that is as representative of a production environment as possible.
3. Make sure to include a review phase for documentation as well, this is also known as static api testing because we are reviewing documentation that is not getting executed
4. Create a small Proof Of Concept by executing 1 API call before going deep into the API and setting all your scripts to work.
5. At this stage we plan our fuzz testing an all other functional testing that has not been identified yet including which tests will be executed at what level
6. Optionally develop stubs and drivers so that testing does not have to wait for integration with a full environment before they can start testing
7. Integrate your API with the full environment and execute your end-to-end testing scenarios
8. Check your test execution results and see if they match the requirements. A failed test case does not automatically mean that the whole feature should be disapproved for production. Careful risk assessment should take place after root cause analysis.
9. Implement any fixes that are required and retest all the failed features
10. When all the requirements have been satisfied, start your User Acceptance Testing (UAT)
11. Optionally, implement any fixes if they are required and retest the failed functionality
12. After all the documentation has been delivered and the requirements have been satisfied we can approve the feature and give it a GO for production
13. Perform a sanity check on the production release before you release it to everyone
14. Follow up on your production releases by checking any logs you have or any potential service desk instances

Test Automation Pyramid



API Testing Best Practices



6. Summary of HTTP Methods

The below table summarises the use of HTTP methods discussed above.

HTTP Method	CRUD	Collection Resource (e.g. /users)	Single Resouce (e.g. /users/123)
POST	Create	201 (Created), ‘Location’ header with link to /users/{id} containing new ID	Avoid using POST on a single resource
GET	Read	200 (OK), list of users. Use pagination, sorting, and filtering to navigate big lists	200 (OK), single user. 404 (Not Found), if ID not found or invalid
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid
PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid

DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution	200 (OK). 404 (Not Found), if ID not found or invalid

Status Codes

1xx Status Codes [Informational]

Status Code	Description
100 Continue	An interim response. Indicates to the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed.
101 Switching Protocol	Sent in response to an Upgrade request header from the client, and indicates the protocol the server is switching to.
102 Processing (WebDAV)	Indicates that the server has received and is processing the request, but no response is available yet.

103 Early Hints	Primarily intended to be used with the `Link` header. It suggests the user agent start preloading the resources while the server prepares a final response.

2xx Status Codes [Success]

Status Code	Description
200 OK	Indicates that the request has succeeded.
201 Created	Indicates that the request has succeeded and a new resource has been created as a result.
202 Accepted	Indicates that the request has been received but not completed yet. It is typically used in log running requests and batch processing.
203 Non-Authoritative Information	Indicates that the returned metainformation in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy. The set presented MAY be a subset or superset of the original version.

204 No Content	The server has fulfilled the request but does not need to return a response body. The server may return the updated meta information.
205 Reset Content	Indicates the client to reset the document which sent this request.
206 Partial Content	It is used when the `Range` header is sent from the client to request only part of a resource.
207 Multi-Status (WebDAV)	An indicator to a client that multiple operations happened, and that the status for each operation can be found in the body of the response.
208 Already Reported (WebDAV)	Allows a client to tell the server that the same resource (with the same binding) was mentioned earlier. It never appears as a true HTTP response code in the status line, and only appears in bodies.
226 IM Used	The server has fulfilled a GET request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.

3xx Status Codes [Redirection]

Status Code	Description

300 Multiple Choices	The request has more than one possible response. The user-agent or user should choose one of them.
301 Moved Permanently	The URL of the requested resource has been changed permanently. The new URL is given by the `Location` header field in the response. This response is cacheable unless indicated otherwise.
302 Found	The URL of the requested resource has been changed temporarily. The new URL is given by the `Location` field in the response. This response is only cacheable if indicated by a `Cache-Control` or `Expires` header field.
303 See Other	The response can be found under a different URI and SHOULD be retrieved using a GET method on that resource.
304 Not Modified	Indicates the client that the response has not been modified, so the client can continue to use the same cached version of the response.
305 Use Proxy (Deprecated)	Indicates that a requested response must be accessed by a proxy.
306 (Unused)	It is a reserved status code and is not used anymore.

307 Temporary Redirect	Indicates the client to get the requested resource at another URI with same method that was used in the prior request. It is similar to `302 Found` with one exception that the same HTTP method will be used that was used in the prior request.
308 Permanent Redirect (experimental)	Indicates that the resource is now permanently located at another URI, specified by the `Location` header. It is similar to `301 Moved Permanently` with one exception that the same HTTP method will be used that was used in the prior request.

4xx Status Codes (Client Error)

Status Code	Description
400 Bad Request	The request could not be understood by the server due to incorrect syntax. The client SHOULD NOT repeat the request without modifications.
401 Unauthorized	Indicates that the request requires user authentication information. The client MAY repeat the request with a suitable Authorization header field

402 Payment Required (Experimental)	Reserved for future use. It is aimed for using in the digital payment systems.
403 Forbidden	Unauthorized request. The client does not have access rights to the content. Unlike 401, the client's identity is known to the server.
404 Not Found	The server can not find the requested resource.
405 Method Not Allowed	The request HTTP method is known by the server but has been disabled and cannot be used for that resource.
406 Not Acceptable	The server doesn't find any content that conforms to the criteria given by the user agent in the `Accept` header sent in the request.
407 Proxy Authentication Required	Indicates that the client must first authenticate itself with the proxy.
408 Request Timeout	Indicates that the server did not receive a complete request from the client within the server's allotted timeout period.

409 Conflict	The request could not be completed due to a conflict with the current state of the resource.
410 Gone	The requested resource is no longer available at the server.
411 Length Required	The server refuses to accept the request without a defined Content- Length. The client MAY repeat the request if it adds a valid `Content-Length` header field.
412 Precondition Failed	The client has indicated preconditions in its headers which the server does not meet.
413 Request Entity Too Large	Request entity is larger than limits defined by server.
414 Request-URI Too Long	The URI requested by the client is longer than the server can interpret.
415 Unsupported Media Type	The media-type in `Content-type` of the request is not supported by the server.

416 Requested Range Not Satisfiable	The range specified by the `Range` header field in the request can't be fulfilled.
417 Expectation Failed	The expectation indicated by the `Expect` request header field can't be met by the server.
418 I'm a teapot (RFC 2324)	It was defined as April's lool joke and is not expected to be implemented by actual HTTP servers. (RFC 2324)
420 Enhance Your Calm (Twitter)	Returned by the Twitter Search and Trends API when the client is being rate limited.
422 Unprocessable Entity (WebDAV)	The server understands the content type and syntax of the request entity, but still server is unable to process the request for some reason.
423 Locked (WebDAV)	The resource that is being accessed is locked.
424 Failed Dependency (WebDAV)	The request failed due to failure of a previous request.

425 Too Early (WebDAV)	Indicates that the server is unwilling to risk processing a request that might be replayed.
426 Upgrade Required	The server refuses to perform the request. The server will process the request after the client upgrades to a different protocol.
428 Precondition Required	The origin server requires the request to be conditional.
429 Too Many Requests	The user has sent too many requests in a given amount of time (“rate limiting”).
431 Request Header Fields Too Large	The server is unwilling to process the request because its header fields are too large.
444 No Response (Nginx)	The Nginx server returns no information to the client and closes the connection.
449 Retry With (Microsoft)	The request should be retried after performing the appropriate action.

450 Blocked by Windows Parental Controls (Microsoft)	Windows Parental Controls are turned on and are blocking access to the given webpage.
451 Unavailable For Legal Reasons	The user-agent requested a resource that cannot legally be provided.
499 Client Closed Request (Nginx)	The connection is closed by the client while HTTP server is processing its request, making the server unable to send the HTTP header back.

5xx Status Codes (Server Error)

Status Code	Description
500 Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
501 Not Implemented	The HTTP method is not supported by the server and cannot be handled.
502 Bad Gateway	The server got an invalid response while working as a gateway to get the response needed to handle the request.

503 Service Unavailable	The server is not ready to handle the request.
504 Gateway Timeout	The server is acting as a gateway and cannot get a response in time for a request.
505 HTTP Version Not Supported (Experimental)	The HTTP version used in the request is not supported by the server.
506 Variant Also Negotiates (Experimental)	Indicates that the server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper endpoint in the negotiation process.
507 Insufficient Storage (WebDAV)	The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request.
508 Loop Detected (WebDAV)	The server detected an infinite loop while processing the request.
510 Not Extended	Further extensions to the request are required for the server to fulfill it.

511 Network Authentication Required	Indicates that the client needs to authenticate to gain network access.

References:
https://katalon.com/api-testing
https://www.wallarm.com/what/what-is-api-testing-benefits-types-how-to-start
https://restfulapi.net/http-methods/