



API Testing

[Introduction](#)

[What is API testing?](#)

[Why API Testing matter](#)

[Benefits of API testing](#)

[Types of API testing](#)

[Unit testing](#)

[Integration Testing](#)

[Performance testing](#)

[Load testing](#)

[Runtime error detection](#)

[Security testing](#)

[Interoperability testing](#)

[Fuzz tests](#)

[Validation Testing](#)

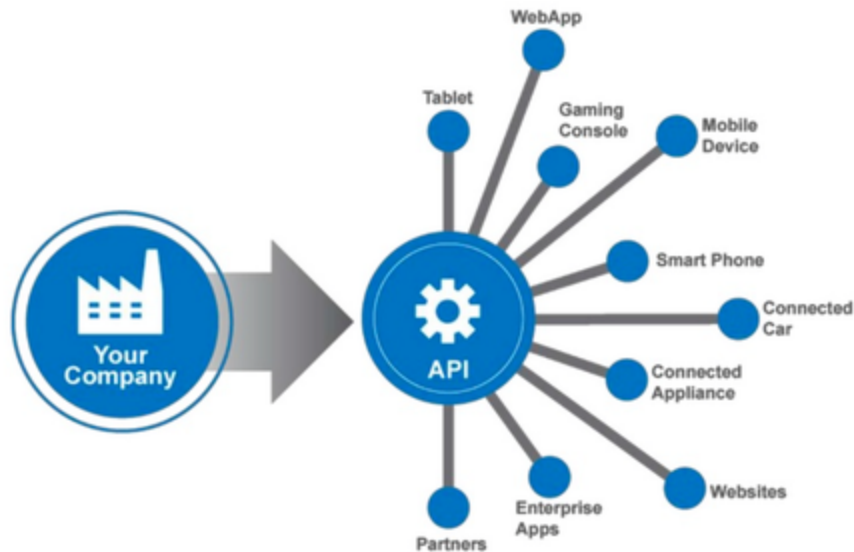
[Manual Testing vs Automated Testing](#)

[How to get started with API testing](#)

[Best Practices of API Testing](#)

Introduction

APIs are becoming very important in our modern world and as technology rises, so will our reliance on APIs. Everything that communicates on the internet these days is talking to an API (Application Programming Interface) and as we implement them in our technologies we also need to take API testing into account. As with everything, our API's need to be tested properly and thoroughly on many different quality characteristics because if we think about it. Not only should we pay close attention to the functional requirements but we should also pay the appropriate amount of attention to non-functional requirements.



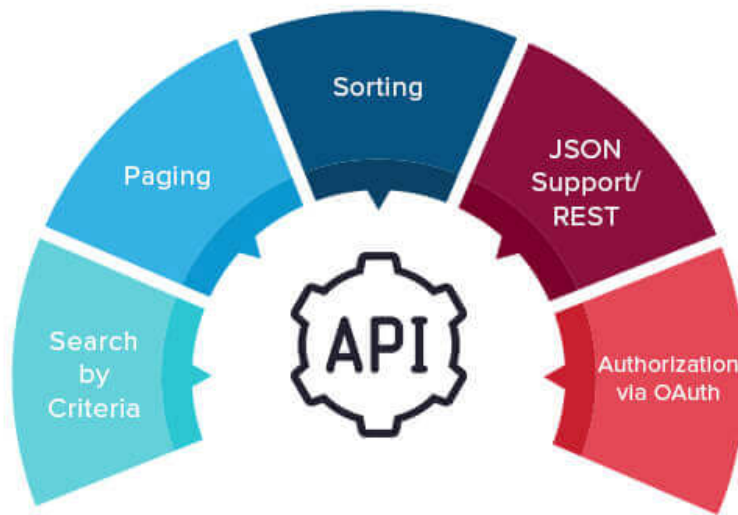
What is API testing?

API testing entails various different aspects and goes much further than simply executing a few test cases. Testing needs to start at the earliest possible stage and continue on until the production release. To highlight these stages, we will be following a fictional feature throughout its software development lifecycle.

We need to start our API tests at the source which would be the requirements. These need to be reviewed very well and from the perspective of a tester. Testers often try to look for edge cases and they will attempt to find ways to break a system. This should be taken into account when reviewing the requirements.



After our requirements have been approved by all stakeholders, test case development can begin. In this stage the testers will start designing their test cases which they will combine into test suites. They also have to make sure to adapt their sanity checks to include this new functionality if that is required by the risk and priority of that new feature.

Features of API



Next we need to determine the test specification which is a document that describes what the expected results are for all the test cases and what the test conditions are under which we can both enter and exit our test, in other words when a test is ready to go and complete. These are also known as our entry and exit criteria.

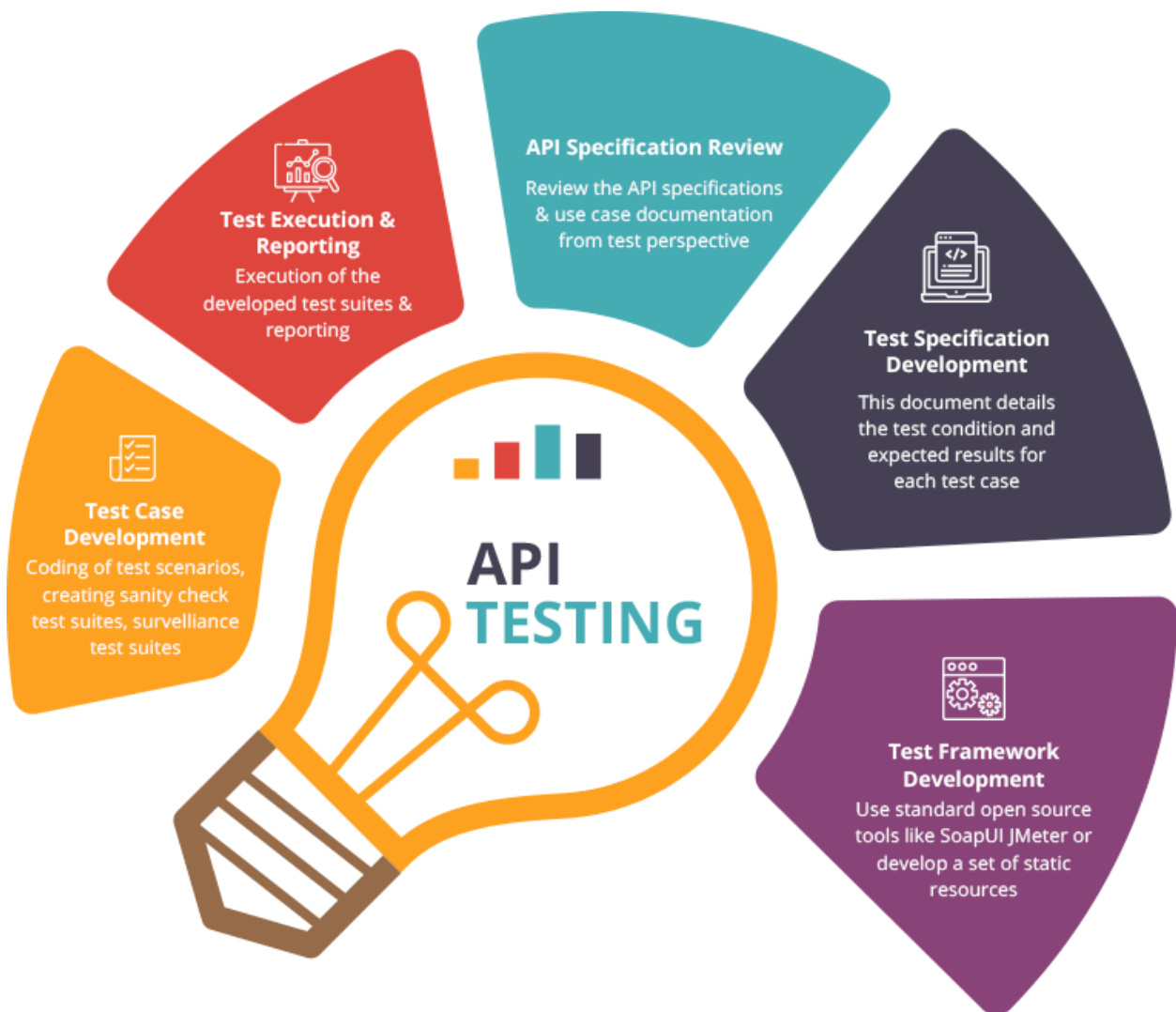
Copy of Entry and exit criteria example

 Entry criteria	 Exit criteria
Integration documentation is available	Unit tests have been created
Test environment is available	Documentation has been created
3rd party integration has been described	95% of the test pass
Untitled	There are no more blocking defects outstanding

When all of the documents have been delivered and the software is ready to be test, the test execution and reporting phase starts. Reports are created based on the test management tools and they will usually include a full report on the tests that have been run for a feature, including the results and any blocking issues encountered which might have halted testing.

Along all of these tests, it is also a good idea to implement a solid test automation strategy which will cut costs drastically if done right and can be achieved using open

source tools such as SoapUI, JMeter, Selenium, postman or any of the other tools out there.



Why API Testing matter

Now you might be wondering why API is so important and that is understandable. After all it costs a lot 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 racks 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

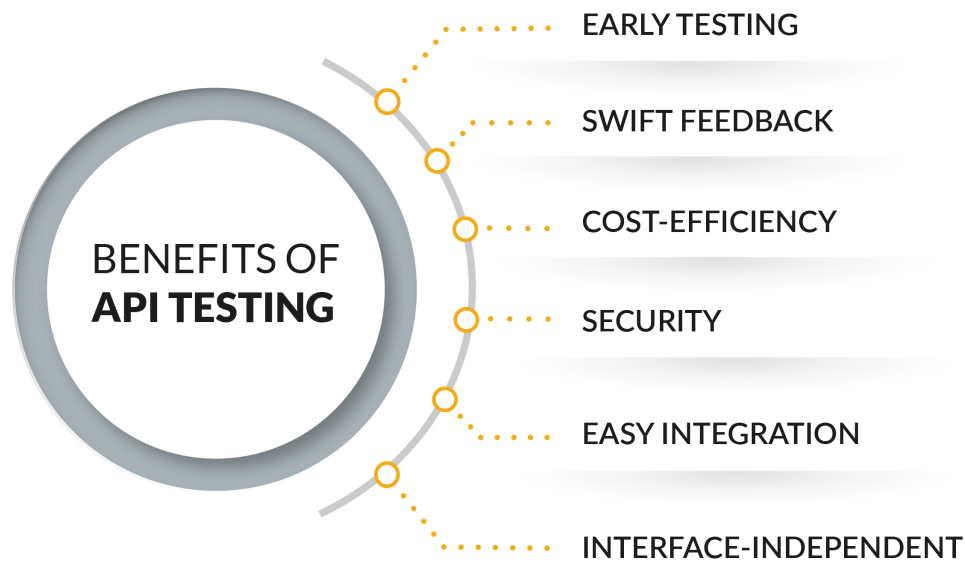
When we perform our API testing, one of the biggest benefits we can enjoy is that we can often test early. This is because the API component is often developed before the UI component which allows for swifter feedback as well and it can help steer development on the components that integrate with our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

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 level 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.



Types of API testing

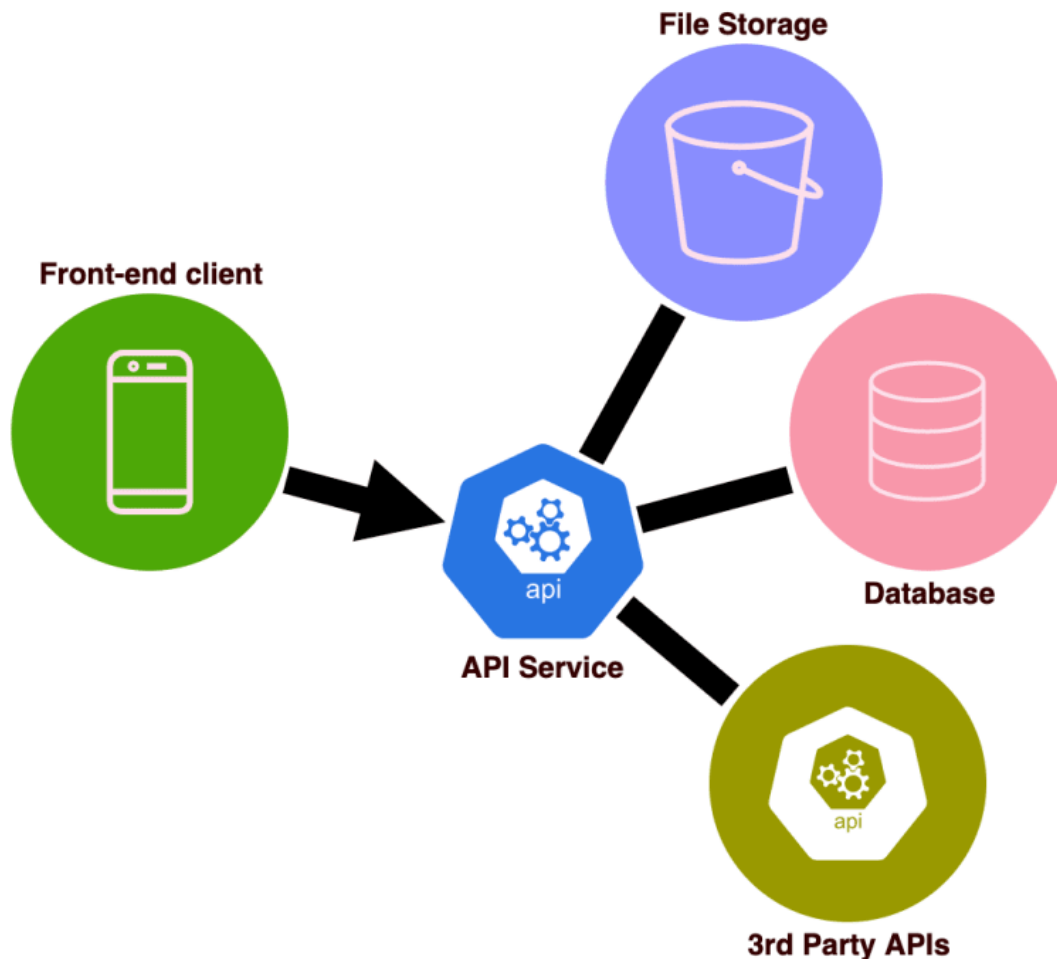
Unit testing

Unit testing are tests that are written to automatically run with every build of the application. They are written close to the code and should pass when running a build of the application. What type of code coverage is required for the APIs depends on the risk the API carries and what functionalities it holds. Good unit testing is like a good foundation and this aspect should be well thought over as it will carry the rest of the testing effort later down the line.

Integration Testing

Our APIs are no separated component of a system. Instead they work to integrate all of the moving parts of a system and all of this integration requires proper testing of course.

Not only do we need to make sure we are passing on and taking in the correct parameters with their correct constraints but we also need to make sure that we sanitise the incoming traffic in a logical way.



Performance testing

Performance testing is one of the non-functional that is most often overlooked and will most often cause trouble. Be very aware that our testing environments often only contain a fraction of the data that production does so any process that runs might take a lot more time in a real scenario. Performance testing should take everything into account including spikes in traffic and multiple heavy processes running at the same

time and interfering with one another. Also be aware that the environment that is used for testing needs to be similar to a production environment to get reliable results.

Load testing

Load testing is very similar to performance testing because we are trying to emulate production like traffic but instead of creating spikes in traffic we want to emulate a constant steady stream of traffic at a normally expected rate. This is to ensure that the API's do not contain memory leaks or other similar defects that might cause issues after running for a prolonged period of time.



Runtime error detection

During all of these tests, we want to make sure that we have Runtime error detection enabled. This technology allows our APIs to report back any defects that occurs while it's in operation.

Security testing

This type of testing is very important but often not budgeted well. We need to ensure that proper security testing occurs based on a risk analysis. This type of testing needs

to occur by a trained professional as it's very easy to miss things but it does not stop there. Every developer should in essence be a little bit of a security tester when it comes to APIs as they are often the first line of defence into our infrastructure.

Security testing has many aliases such as pentesting, PEN testing or penetration testing and needs to focus on several aspects when it comes to api testing. the entry points of our APIs needs to be taken into account but also the flow of data and any shadow APIs that are no longer in user but are still in operation.

Interoperability testing

Working with 3rd party software providers or even older versions of our software is not something we can take for granted. We need to create a section in our test plan where we describe what testing will be done. Severity and priority of potentials defects need to be discussed based on prior experiences and they need to help define how API testing occurs when the APIs interact with

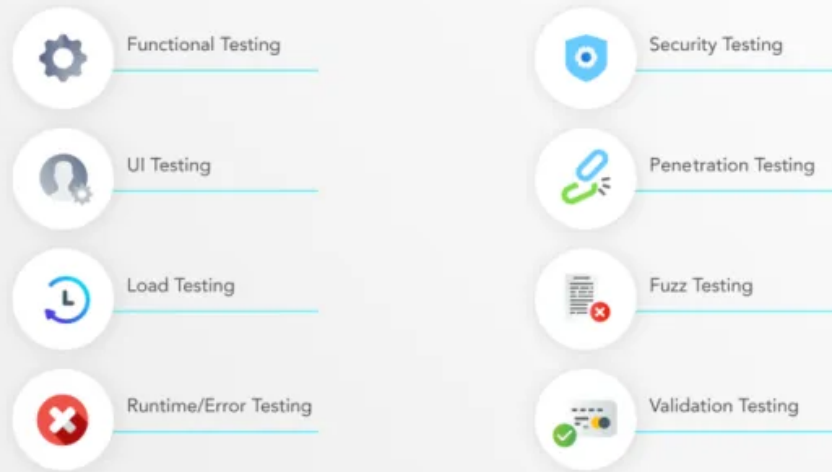
Fuzz tests

As a final test before we validate our application we need to fuzz all the endpoints of our APIs. When fuzzing we will send random data to those API endpoints and we need to carefully inspect the results. Our server should not crash from this unexpected traffic and it should not display any odd behaviour. Based on a risk analysis, fuzz testing might be performed much more structured or not at all.

Validation Testing

During validation testing we need to ensures that the software meets the business requirements. The testers need to evaluate if the test execution results match what is expected and required by the test plan. They will subsequently guide the business users in their User Acceptance Testing (UAT) in which they ensure the stakeholders execute the pre-built testing scenario's and report any deviations on what was expected. If the software is approved

API testing types



Manual Testing vs Automated Testing

When thinking about automated testing, we need to take into account that it's expensive to create these tests and maintain them. After all, when the flow of the application changes, our API call order or the parameters might not be set correctly anymore. Every change like this requires changes in our test automation framework and even setting up the framework might be a big investment. You will find however that often this investment is more than worth it if you have to run a suite of tests every day or every release to the test environment. As a general rule of thumb I try to think that something is worth automating if you have to test it at least 10 times.

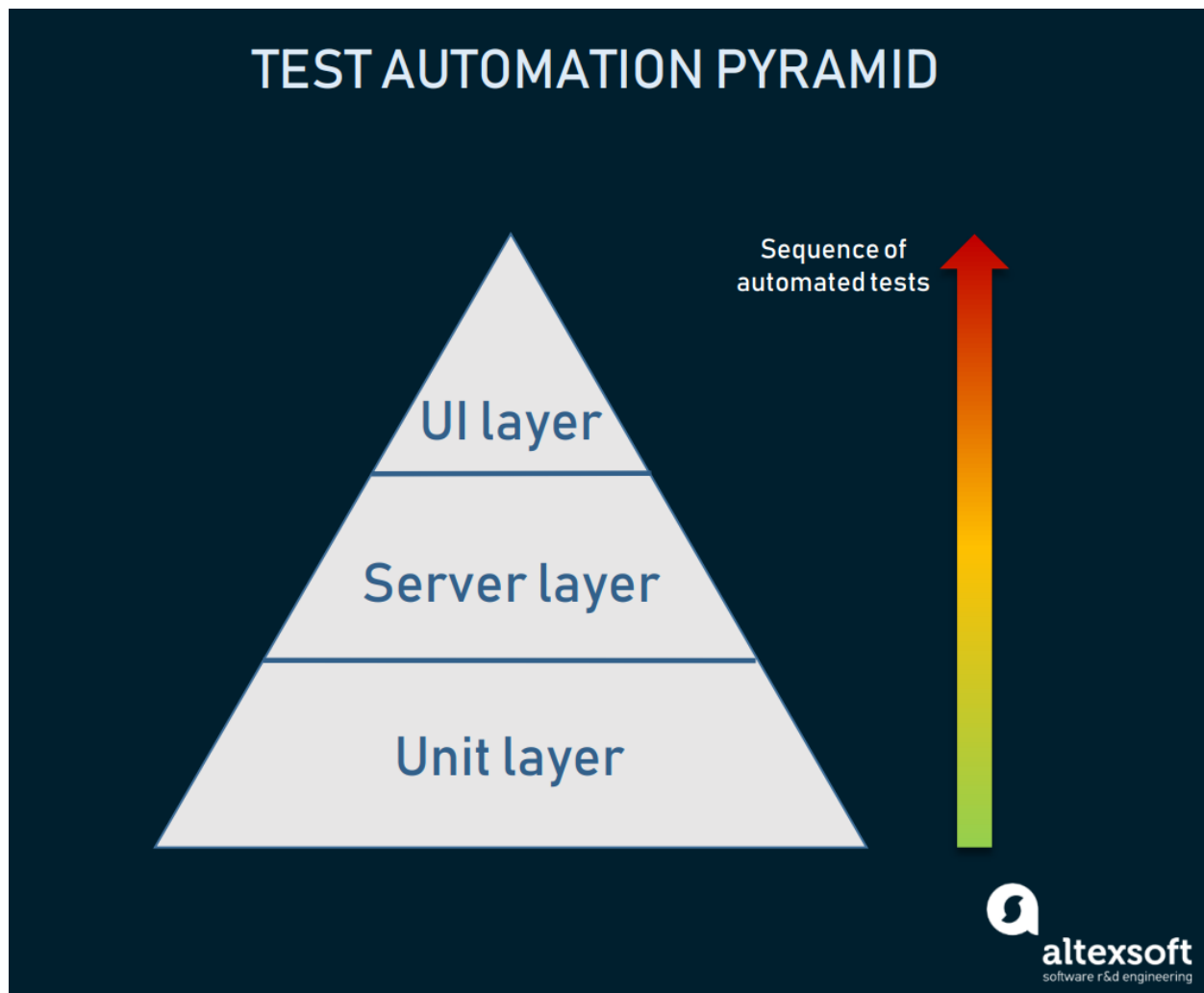
API test automation has many advantages as well. We can write test scenarios very quickly so we can easily create test cases for a lot of edge cases. We also don't have to run the tests manually so much and we can include our build pipelines. If a build produces too many errors at that moment, we want to be informed before we even start testing as a new build will be inevitable after fixing the errors that were found.



How to get started with API testing

- 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 needs to be answered to which no single human can have the answer. This is why team work matters, developers, testers, system architects, analysts, product owners... they all know their domain best and should be included in the creation of requiremensts.
 - 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
- Next we need to set up a good testing environment that is as representative of a production environment as possible.

- 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
- Create a small Proof Of Concept by executing 1 API call before going deep into the API and setting all your scripts to work.
- At this stage we plan our fuzz testing and all other functional testing that has not been identified yet including which tests will be executed at what level
- Optionally develop stubs and drivers so that testing does not have to wait for integration with a full environment before they can start testing
- Integrate your API with the full environment and execute your end-to-end testing scenario's
- 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.
- Implement any fixes that are required and retest all the failed features
- When all the requirements have been satisfied, start your User Acceptance Testing (UAT)
- Optionally, implement any fixes if they are required and retest the failed functionality
- 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
- Perform a sanity check on the production release before you release it to everyone
- Follow up on your production releases by checking any logs you have or any potential service desk instances



Best Practices of API Testing

- It is important we group our test cases well by functionality. This will improve maintainability and will give a better overview of the status of the application
- Make sure you include the API's workflow, how it fits in the application and the edge cases into the requirements and make sure all stakeholders are involved in the creation of these requirements
- We need a testing environment that closely resembles production to test API requests against. While it's possible to test against stubs and drivers to start development or speed it up but we need to make sure to always test end-to-end scenario's on a sandbox environment.

- Follow the testing triangle carefully. This means that you automate what is not possible to be tested via unit or integration tests. Automation is a big cost saver and can detect serious issues early on if implemented correctly

