Game changer for API Testing: KushoAI

What is KushoAI?

KushoAI is an **AI Agent for API testing** that generates **exhaustive test suites for your APIs in minutes** – all you need to do is input your API information and sit back while KushoAI figures out what real-world scenarios can occur for your API in production and write ready-to-execute tests for them.

How KushoAIwork?

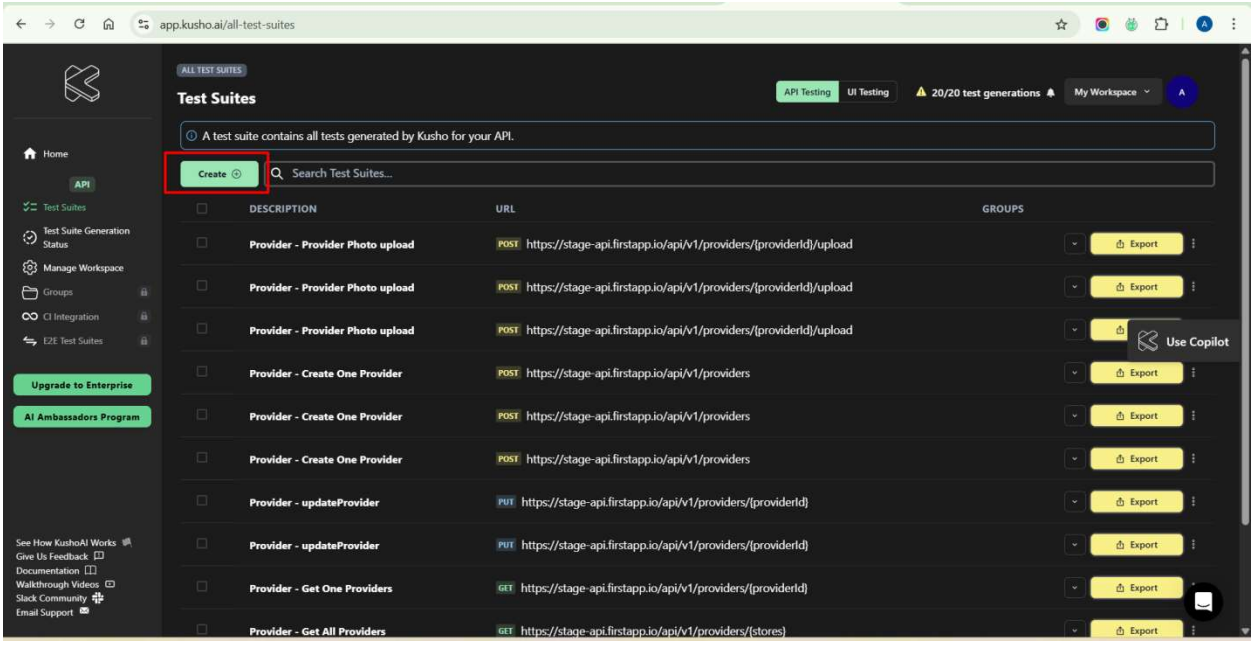
KushoAI is designed to work the same way a Dev or QA works while testing an API. You can tell KushoAI about your API by uploading a spec (OpenAPI, Postman Collection, cURL, RAML and many more to come) or by entering information about your API manually.

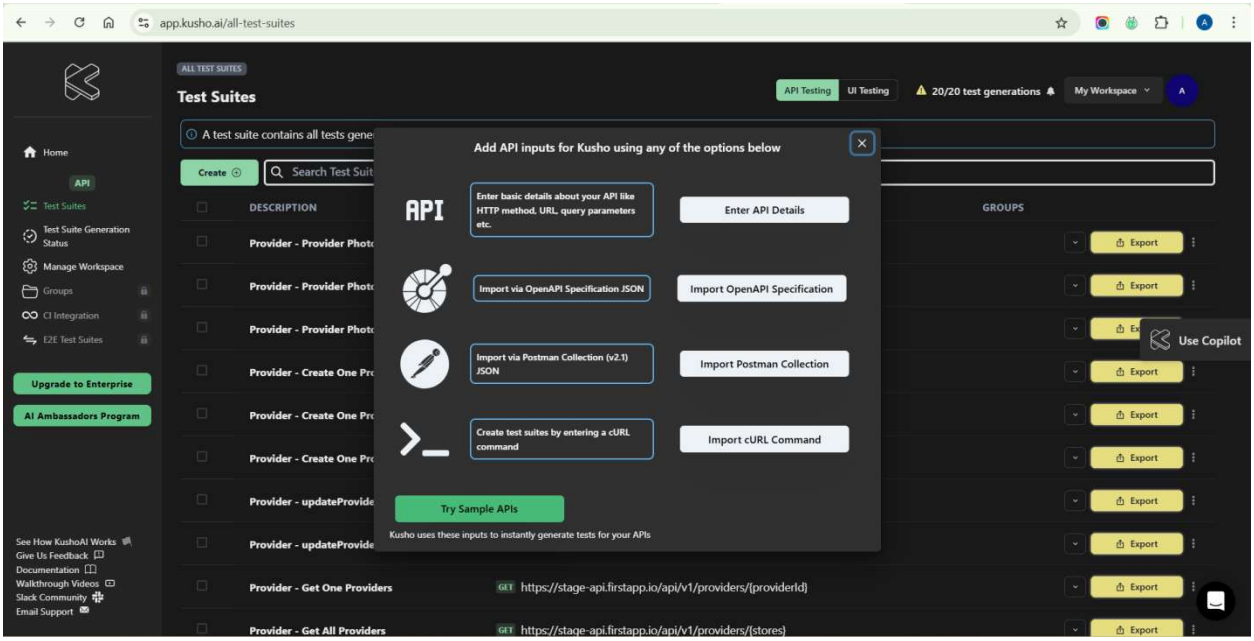
After that, KushoAI analyses different aspects of your API and its payload, which ranges from simple things like what's the data type of fields to complex stuff like semantics and the format of the value. If you're using specs to provide API information, KushoAI will also pick up cues from the field and API description that you've written in your specs. Based on this, KushoAIgenerates scenarios that can occur for your API in production and creates tests that you can execute directly from the KushoAI web app.

Different ways to create a test suite in KushoAI:

To create a test suite, click on the "Create" button on Test Suites page and pick a format for sharing your API information. You can input your API information into KushoAI in four different ways:

1. Manually enter API details
2. Import OpenAPI specs
3. Import Postman Collections
4. Copy and paste cURL command

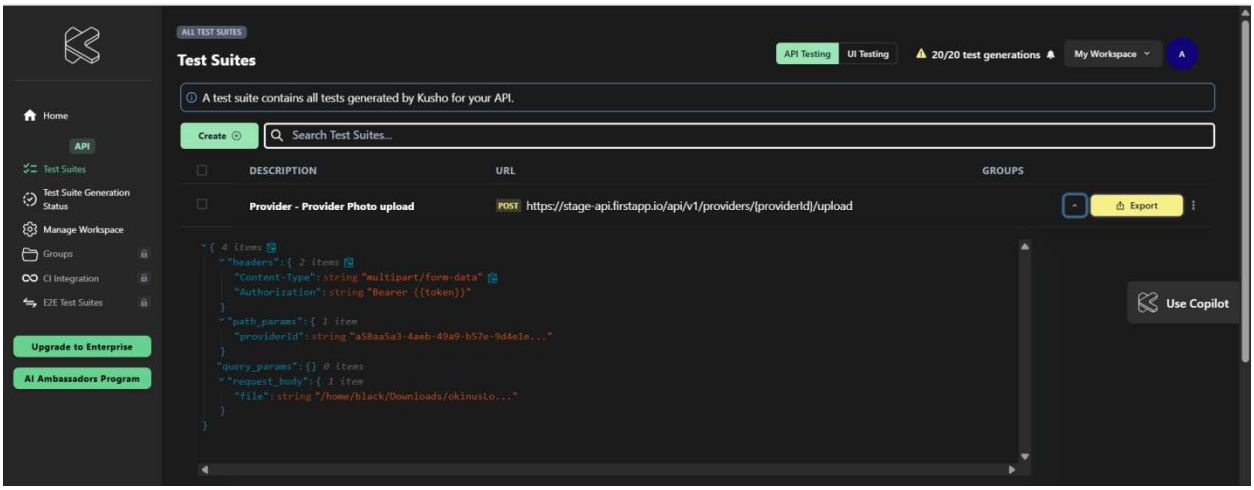




Test Suites Page:

You can view all the test suites created to date on the Test Suites page.

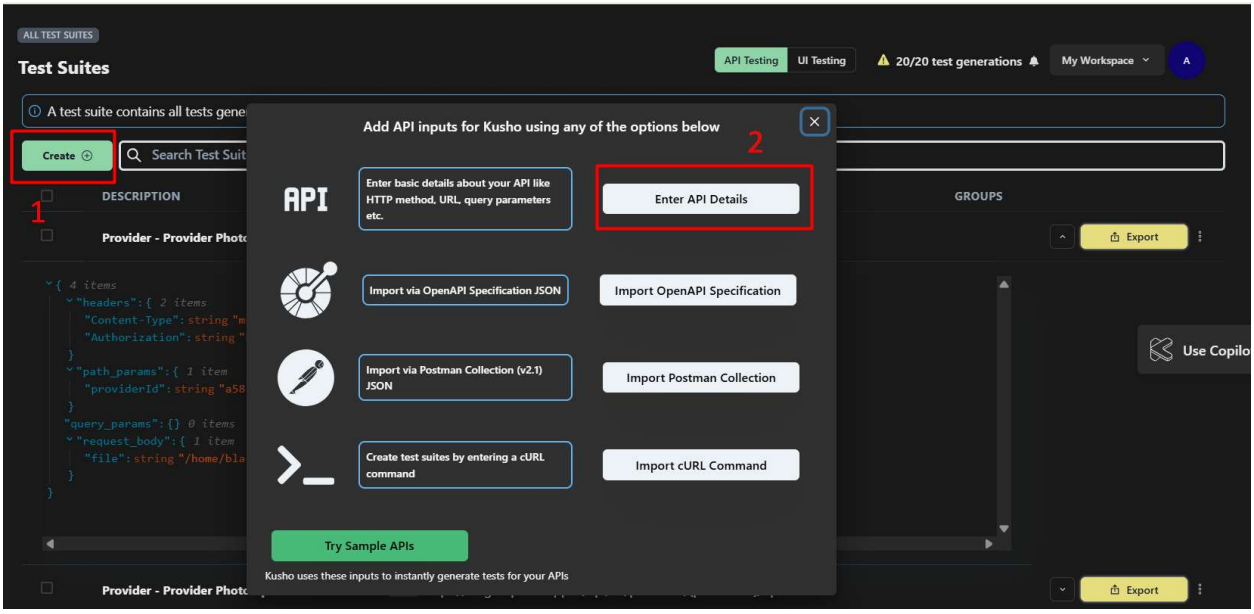
Clicking the down arrow will expand the test suite row to show details about the API.



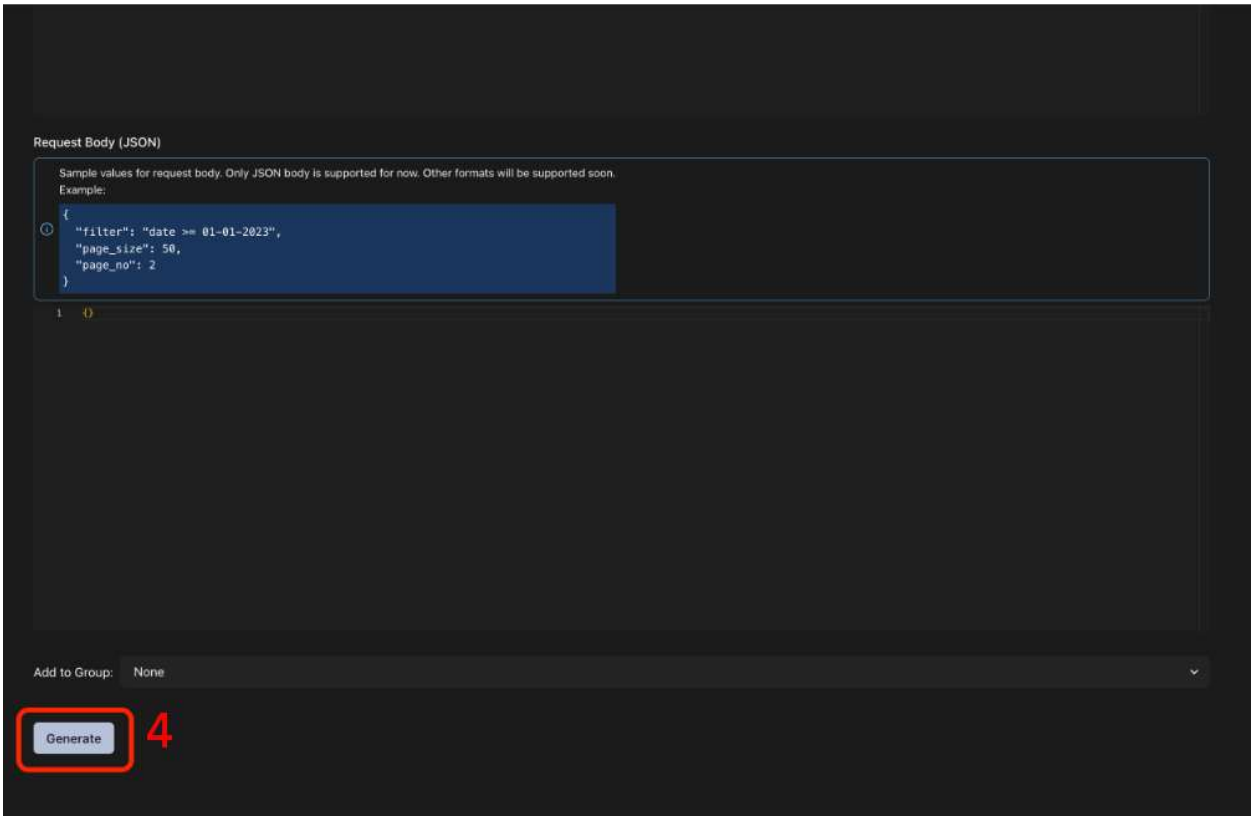
1. Manually enter API details

If you're building a backend application and don't have it all documented yet, you can just manually enter your API information.

1. On the Test Suites page, click on the "Create" button
2. Click on the "Enter API Details" button



3. Enter basic details about your API, like URL, headers, path/query params, and request body here
4. Click on "Generate" to start test generation. This typically takes 3-4 minutes based on how big the API payload is.

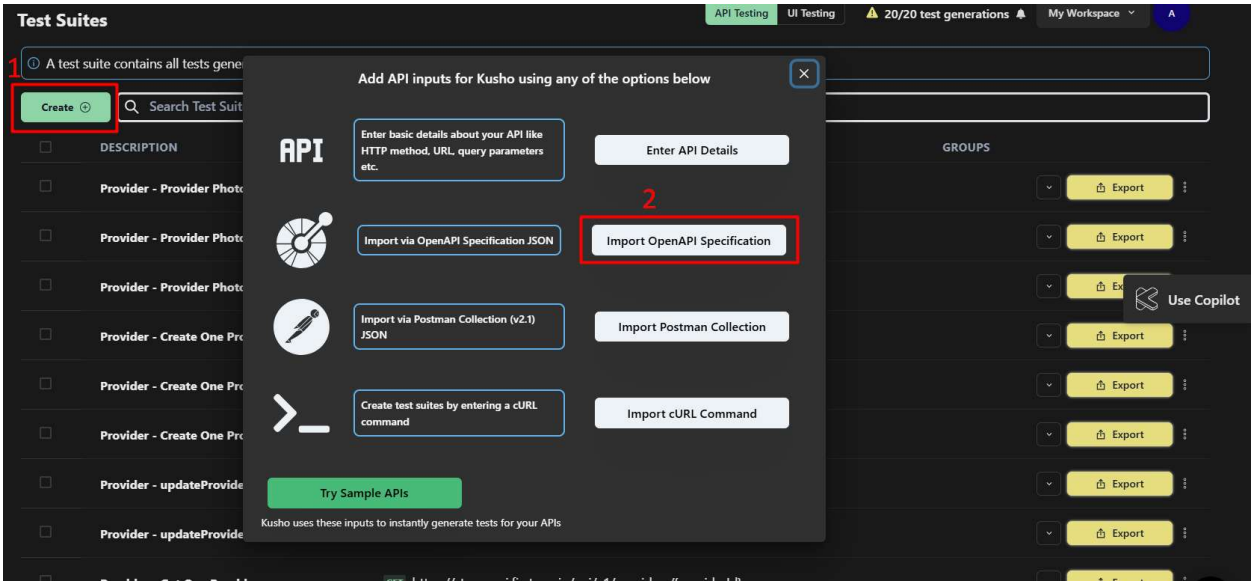


2. Import from OpenAPI specs:

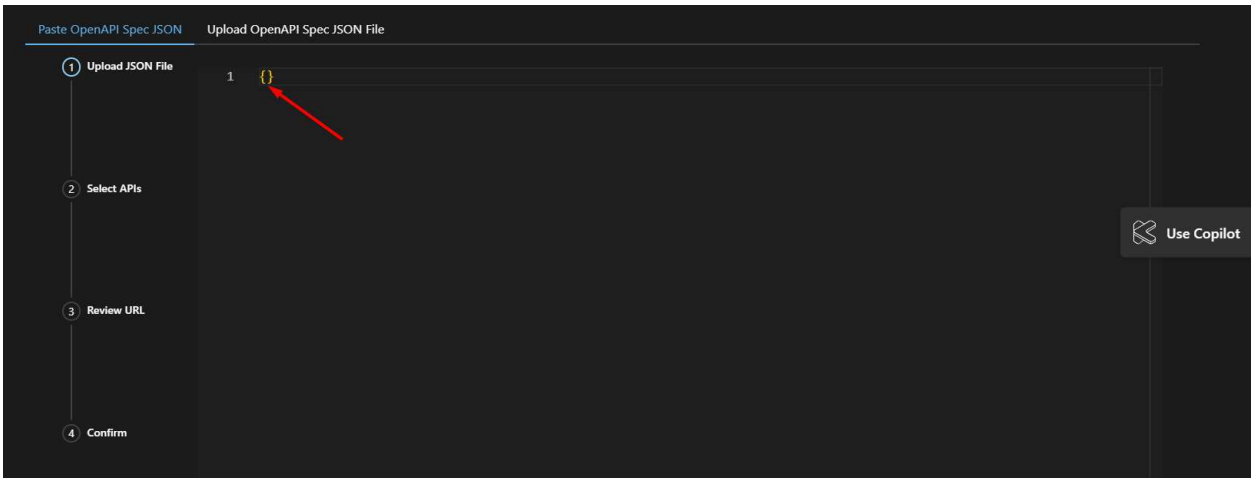
If you use Swagger and have an OpenAPI spec available for your APIs, you can use this spec to provide API information to KushoAI and get started. Using OpenAPI spec allows you to import multiple APIs into KushoAI in one go and provide additional information about your APIs (datatypes of fields, range of values, optional/compulsory, etc.).

Here's how you can use an OpenAPI spec to create a test suite:

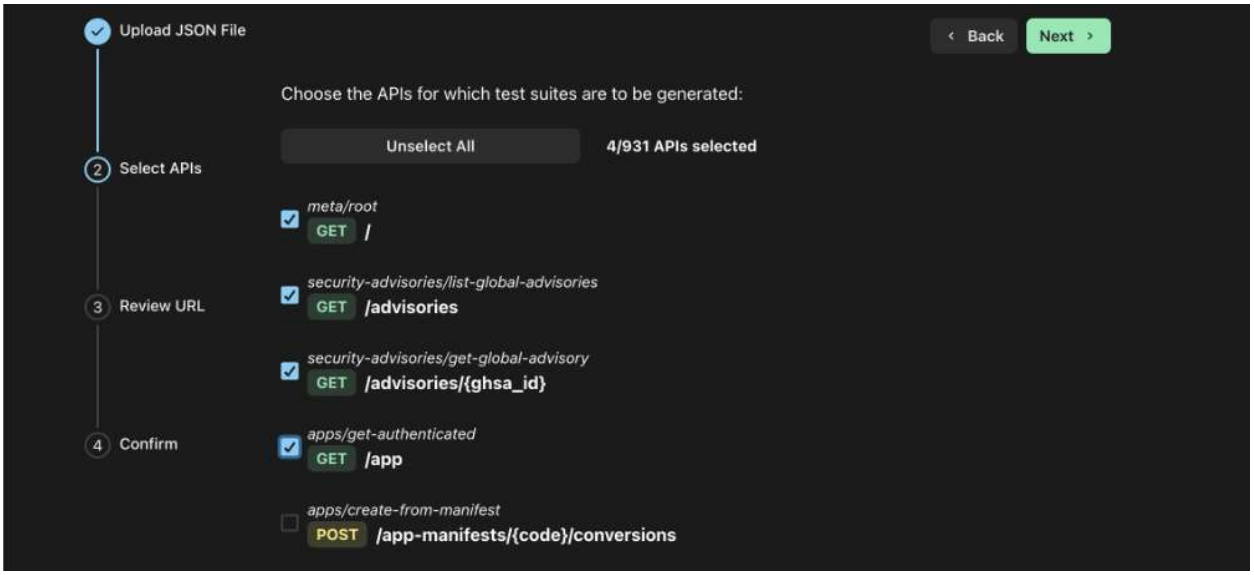
1. On the Test Suites page, click on the "Create" button
2. Click on the "Import OpenAPI Specification" button



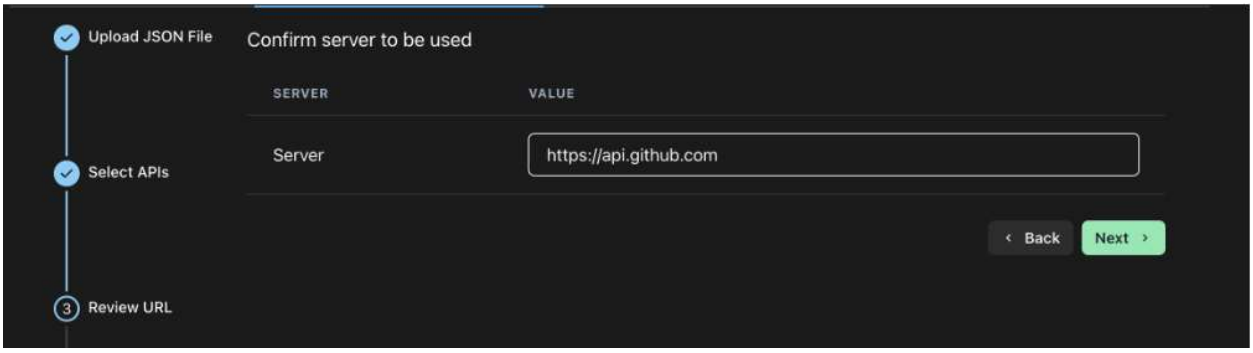
3. Paste your OpenAPI spec or upload the JSON file



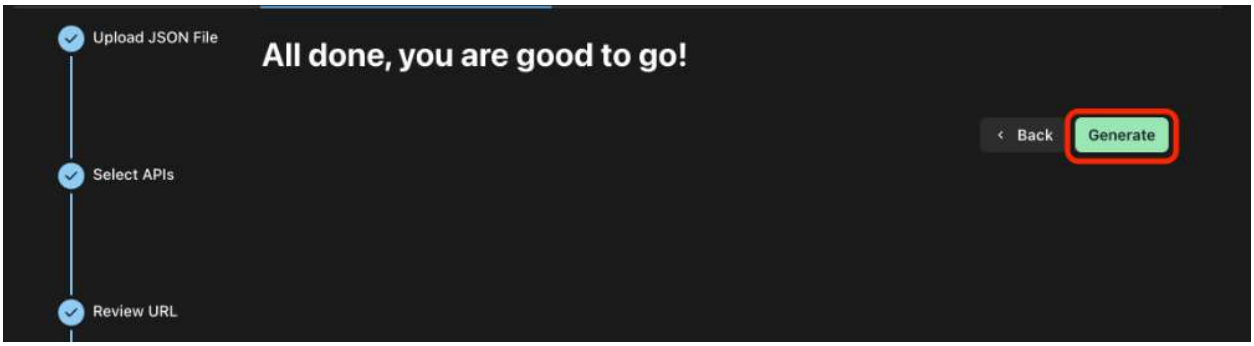
4. Select the APIs that you wish to import



5. Enter the server URL you wish to use for testing. Don't worry about having to stick to a single URL. You can use environments and variables to set up as many different URLs as you want



6. Click on "Generate" to start the test generation



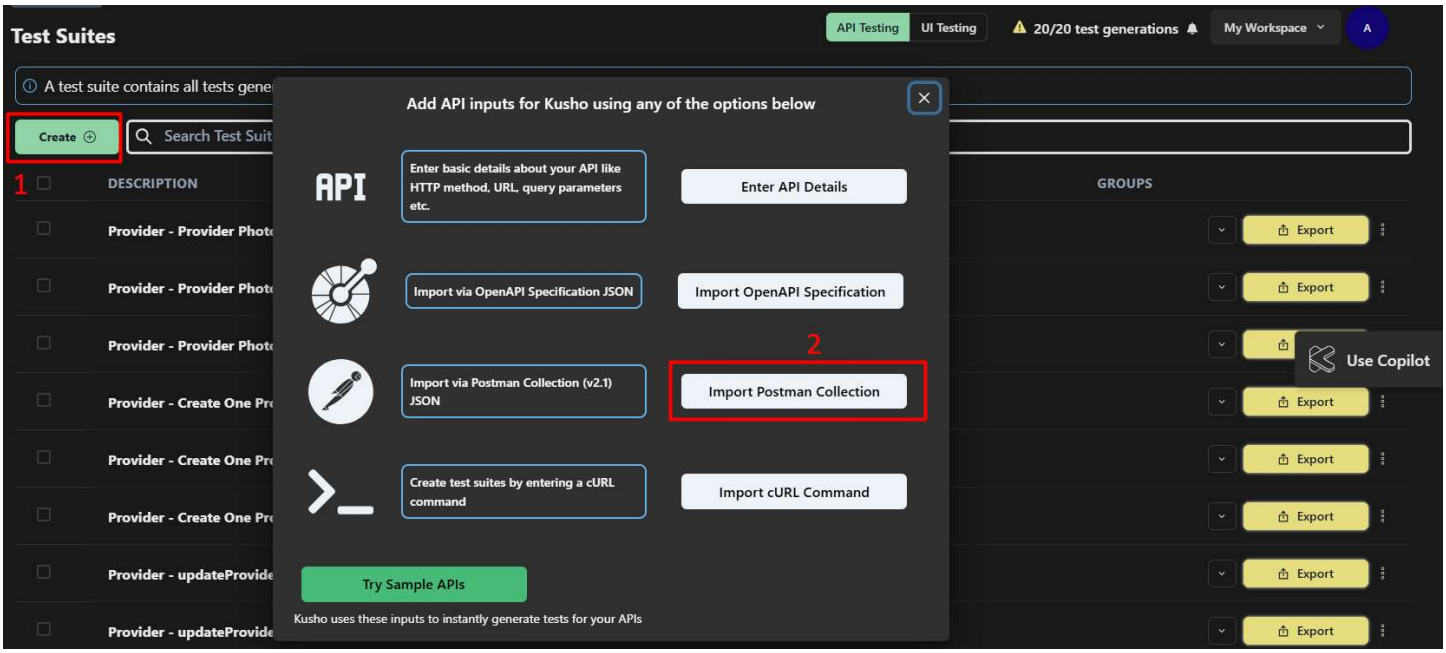
7. Generation using OpenAPI spec happens in async. You'll receive an email once the generation is completed, after which you can view your test suites on the Test Suites page. In the meantime, you can use the Test Suite Generation Status page to check the status of your APIs.

3. Import from Postman Collections:

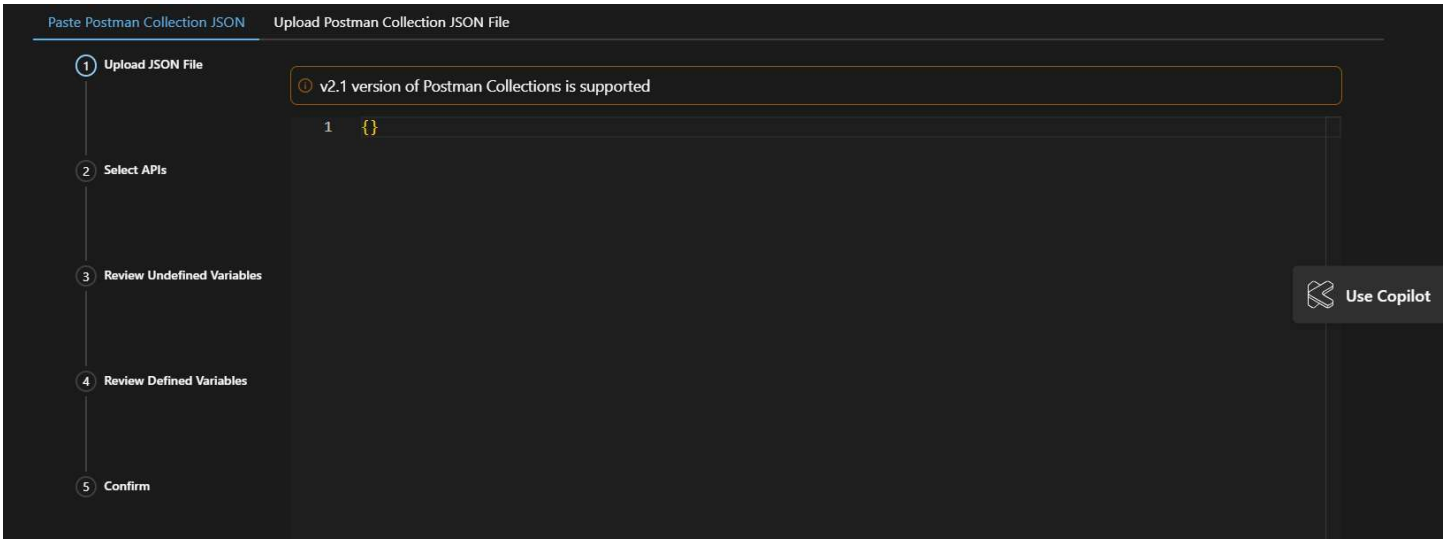
If you have your APIs on Postman, you can use the Postman collection option to bring them to KushoAI. Importing the Postman collection allows you to import multiple APIs into KushoAI in one go.

Here's how it works:

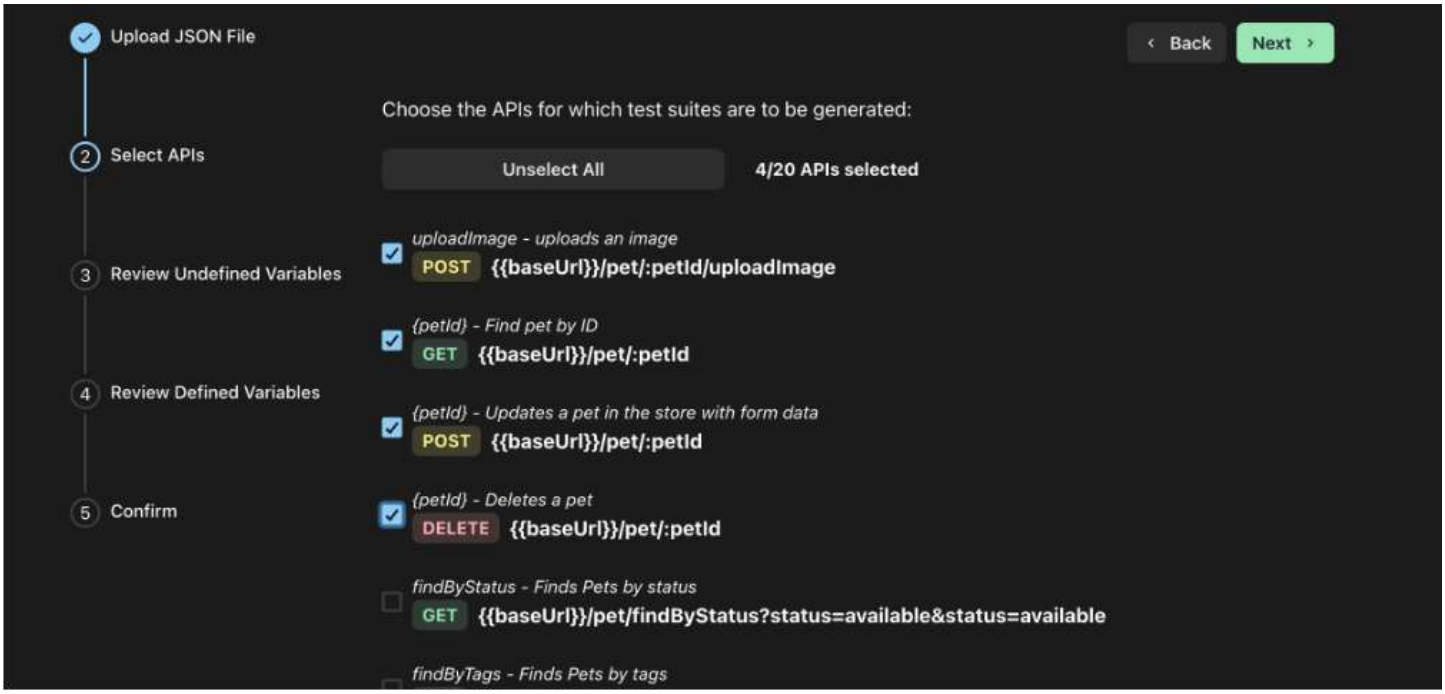
1. On the Test Suites page, click on the "Create" button
2. Click on the "Import Postman Collection" button



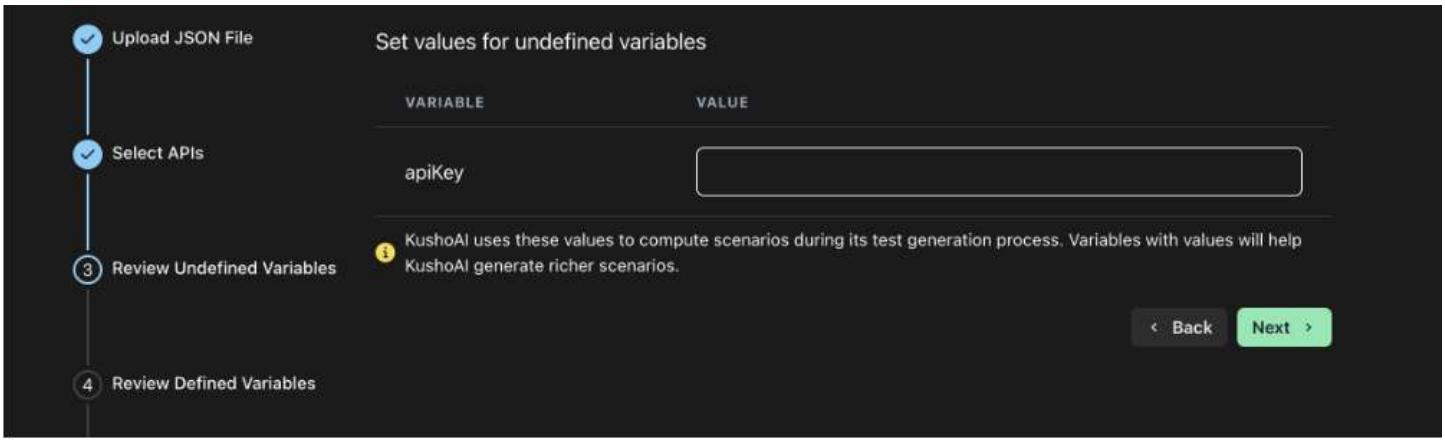
3. Export the collection JSON form Postman
4. Paste your collection JSON or upload the JSON file

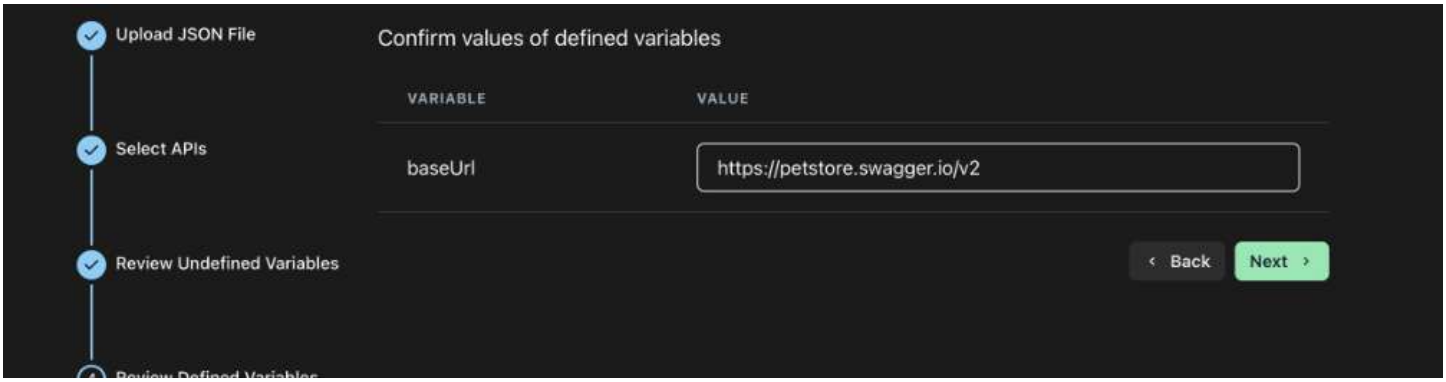


5. Select the APIs that you wish to import

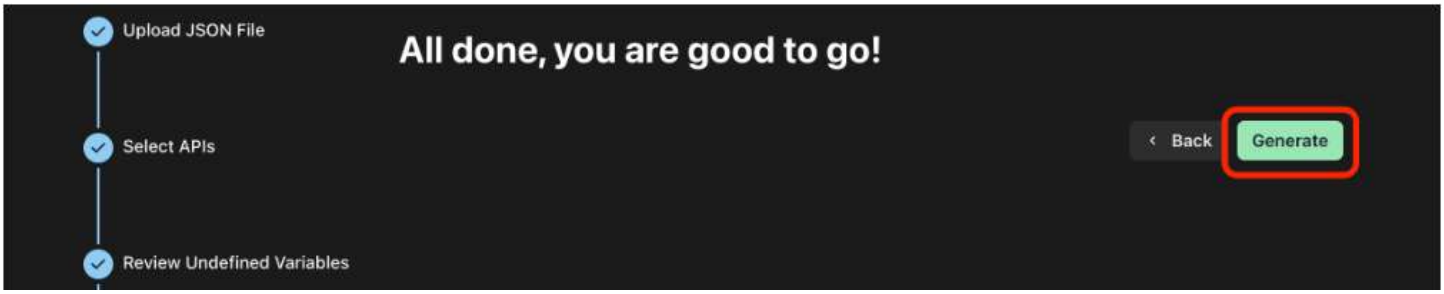


6. If you're using variables, provide/review their values. #TODO: Explain why a user needs to provide their values when they're already present on Postman.





7. Click on "Generate" to start the test generation

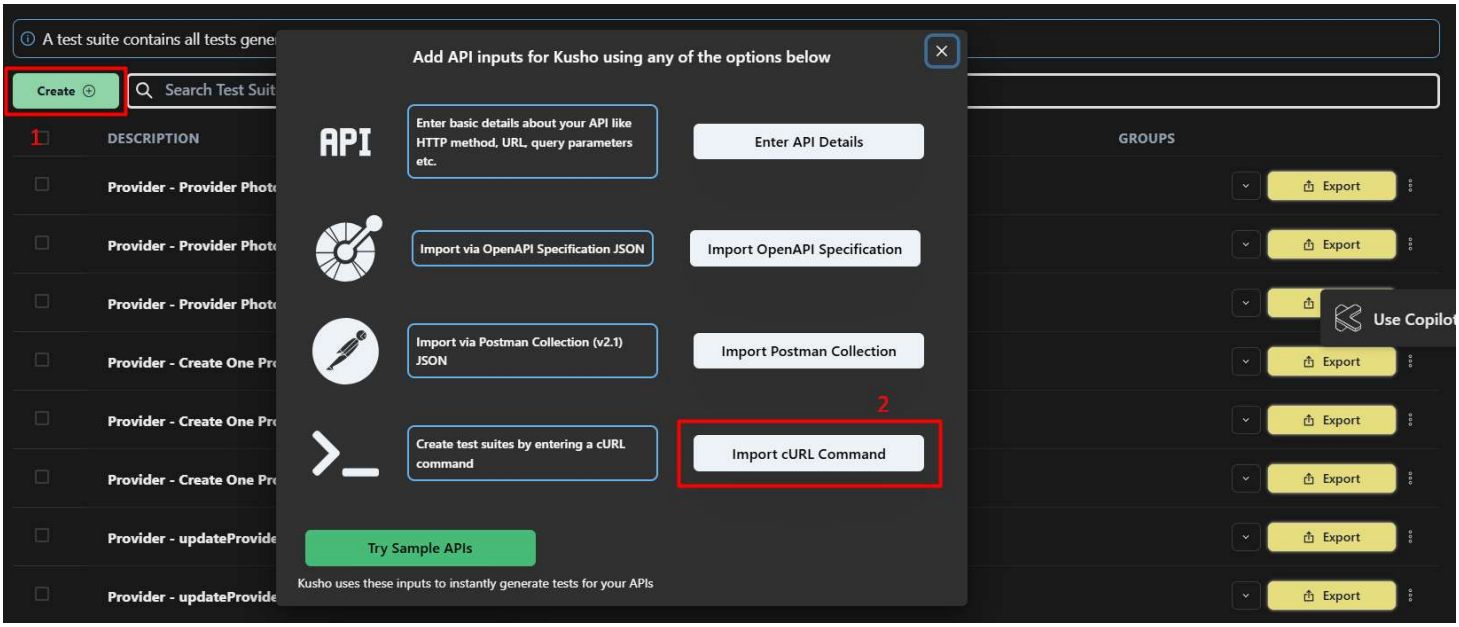


8. Generation using Postman collection happens in async. You'll receive an email once the generation is completed, after which you can view your test suites on the Test Suites page. In the meantime, you can use the Test Suite Generation Status page to check the status of your APIs.

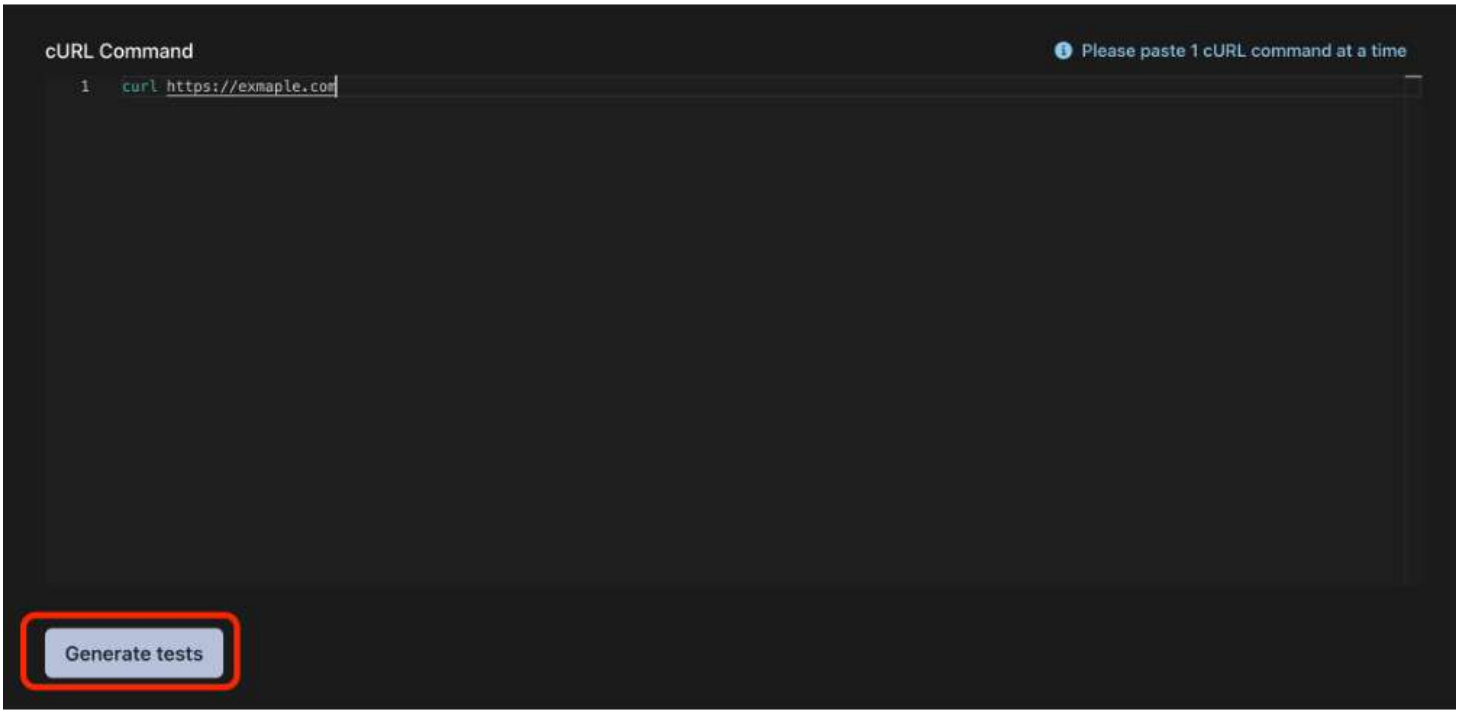
4. Import using cURL:

If you're an OG 👴 and use cURL for testing, you can add your API information using cURL commands like this:

1. On the Test Suites page, click on the "Create" button



2. Paste your cURL command here. Please note that they process one cURL at a time.
3. Click on “Generate” to start the test generation



4. Generation using cURL command happens in async. You'll receive an email once the generation is completed after which you can view your test suites on the Test Suites page. In the meantime, you can use the Test Suite Generation Status page to check the status of your APIs

Test suite details:

Once you're done creating a test suite for your API, clicking on a test suite row on Test Suites page will take you to the Test Suite Details page. All the tests generated by KushoAI for your API are displayed on this page, along with a bunch of operations that you can perform on them.

Let's take a look at what's present on this page and how you can use the operations available here to test your API.

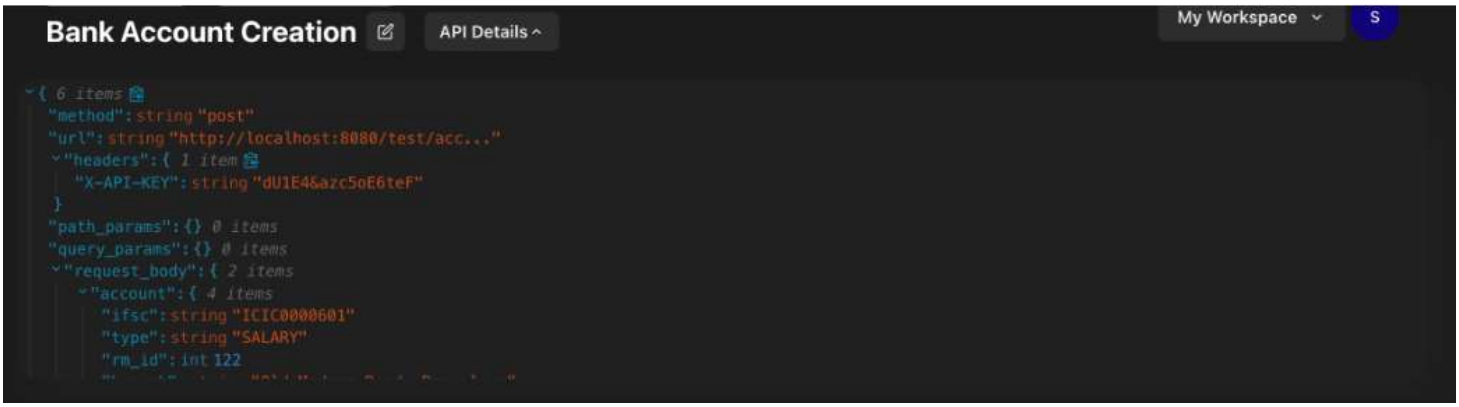
Test suite name:

At the top of the page is the name of your test suite. This name is generally derieved from your spec/collection name concatenated with the name of your API. In case of test suites which were created by manually entering API information or using cURL command, the name is auto-generated. In case you want to rename the test suite, you can do so by clicking on the pencil icon next to the name.

Test suite API information:

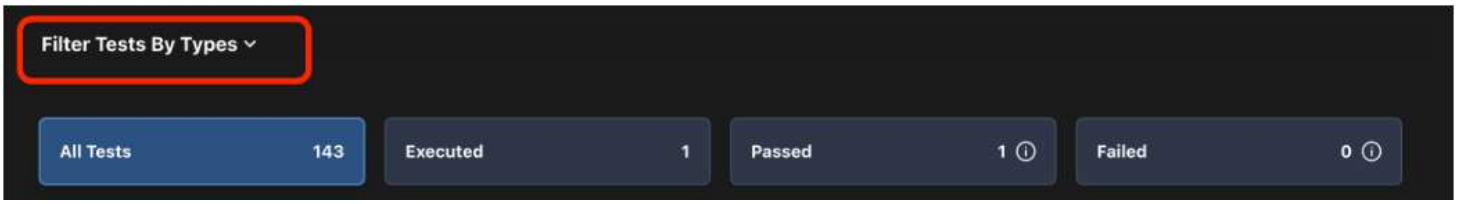
The "API Details" button displays the API information for a test suite. This is the information that we picked up from a spec/collection or was entered by you manually during the creation of the test suite. During test

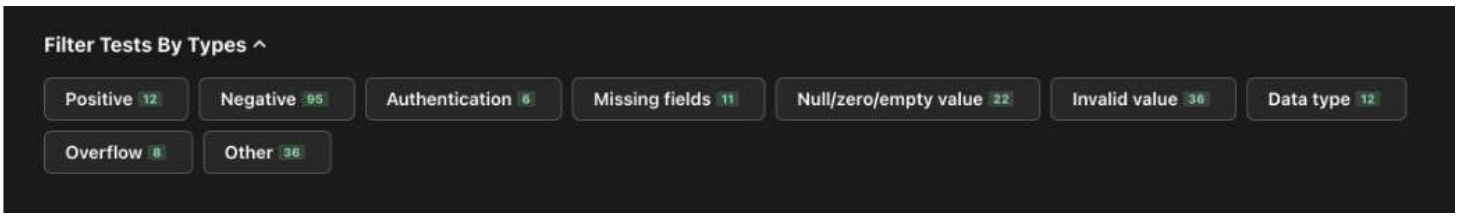
generation, KushoAI uses this API information to create payloads for each test. This API information is editable, and changing this will trigger regeneration of the tests in a test suite. More details about this here.



Filter tests by types:

KushoAI generates tests belonging to various categories. Click on "Filter Tests By Types" accordion to view all the different categories for which KushoAI has generated tests. You can also use the categories here to filter tests based on their categories.

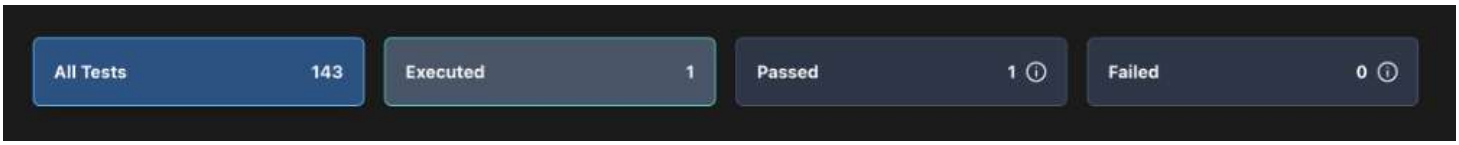




Test counts:

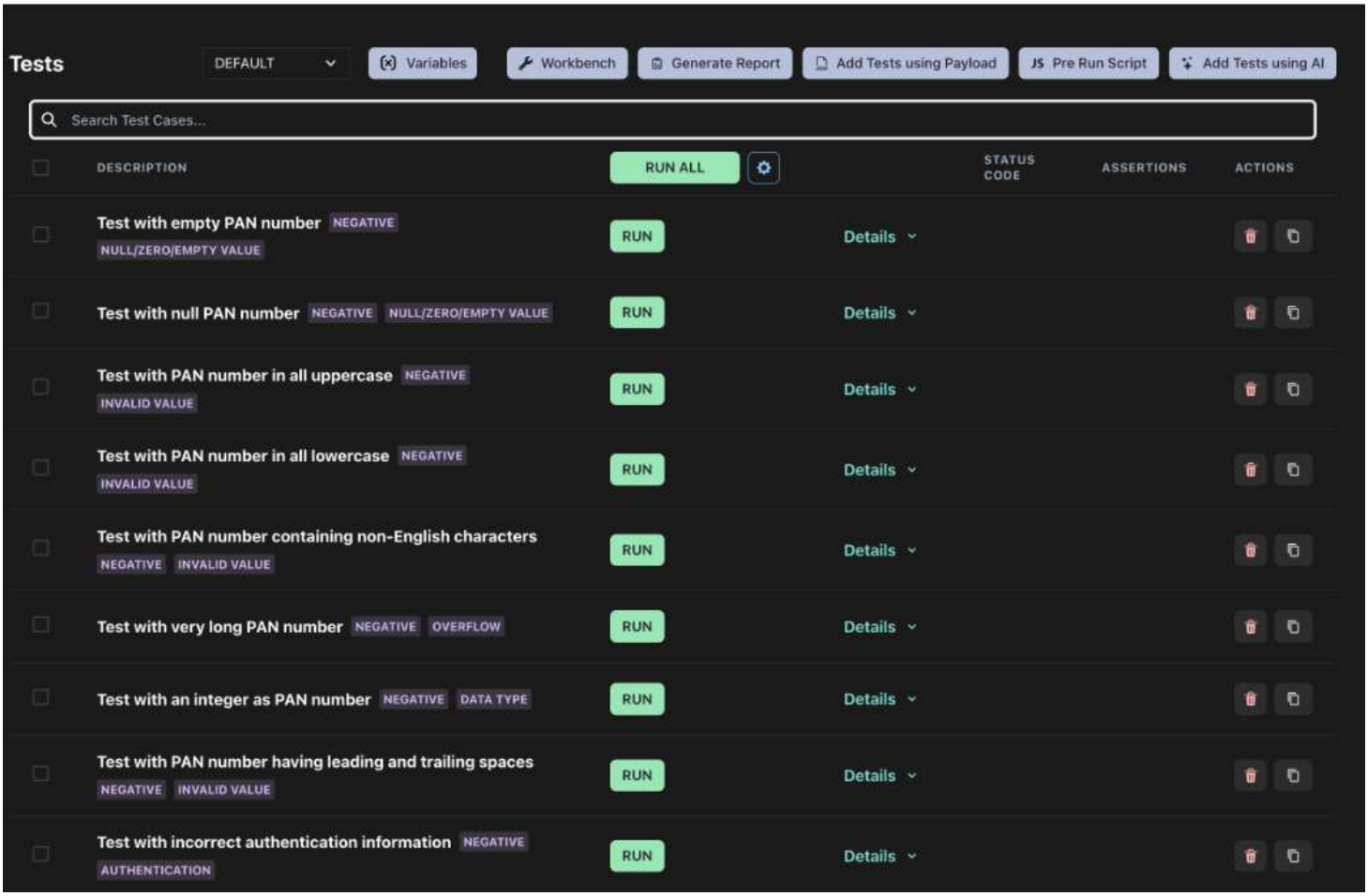
The test counts section, as the name suggests, displays counts of tests in various states

- All tests: This is the total number of tests present in a test suite
- Executed: This is the number of tests executed using any of the available run options
- Passed/Failed: This is the number of tests marked as passed/failed based on assertions

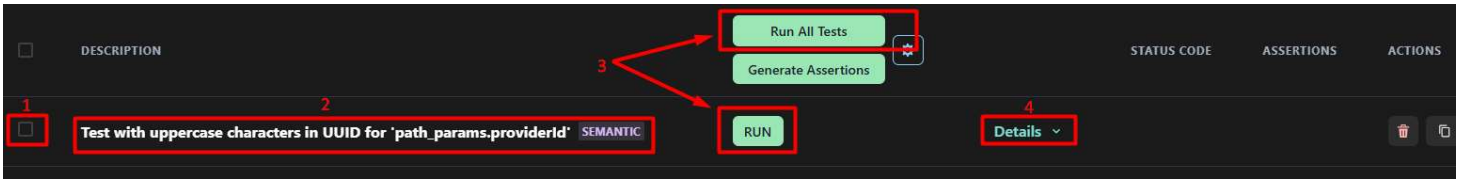


Tests Table:

This is the most important section of this page. This is where all the tests generated by KushoAI (and later added by you) are shown. Let's take a look at each column of this table:

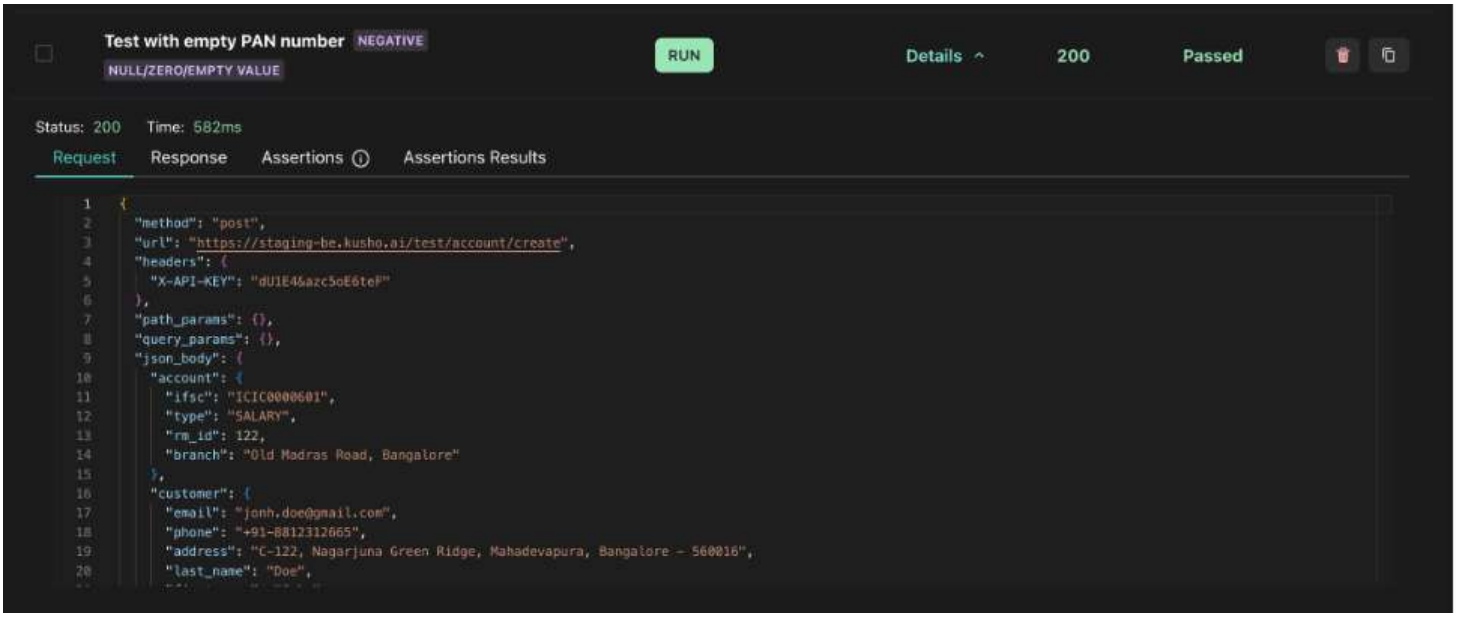


1. **Checkbox -** This can be used for selecting multiple tests and performing actions on them like running, adding/removing tags, etc.
2. **Test description -** This is a high-level plain English description of the objective of the test (e.g. Test without proper auth token, Test with null value for name field). Next to the description, you'll see the categories and tags for the test.
3. **Run -** This column has the button for running a test. The column header has a "Run All" button, which can be used to execute all tests in one go. The gear icon next to "Run All" button can be used to configure running-related parameters like wait time, parallelization, etc.
4. **Details -** Clicking this button will open a drop-downthat displays the ***Test Details*** section with the following information:



   a. **Status and Time:** After running a test, the status code received in the API response and time taken by the API to execute will be displayed here.

b. **Request:** This is the API payload for the test. This is a variation of the API information that was imported/entered during test generation based on what's being tested. This payload is editable.
c. **Response:** After running a test, the API response will be displayed here
d. **Assertions:** Assertions are JS code used for validating the API response to determine if a test has passed or failed (similar to asserts in unit tests)
e. **Assertion Results:** This is where results for each assertion will be displayed after running a test



5. **Status Code -** After running a test, the status code in API response is displayed here
6. **Assertions -** Pass/fail status of the test is displayed here after running a test. Note that if assertions are not present, this will be blank or "N/A"
7. **Actions -** Actions that can be performed on a test (like delete, copy UUID for a test, etc.) are displayed here.

Test Suite Toolbar:

This section displays a bunch of operations that can be performed on a test suite. Let's go over what these operations are:

1. **Environment selector:** Use this switch between environments. Environments allow you to set up different sets of variables (like URL, auth tokens, etc.) for each environment (e.g. localhost, qa, stage, pre-prod, prod) that you plan to use for testing.

2. **Variables:** Variables can be used to store values that keep changing (e.g. auth tokens, API keys, etc.). Use this to add/edit/delete variables.

3. **Workbench:** You can use this feature to perform bulk operations on tests like editing API payloads, generating and editing assertions for all tests in a single pane.

4. **Generate Report:** Reports provide a high-level summary of a test run. Use this button to send a report in an email or to download a report.

5. **Add Tests using Payload:** Use this to add more tests to the test suite by defining API payload.

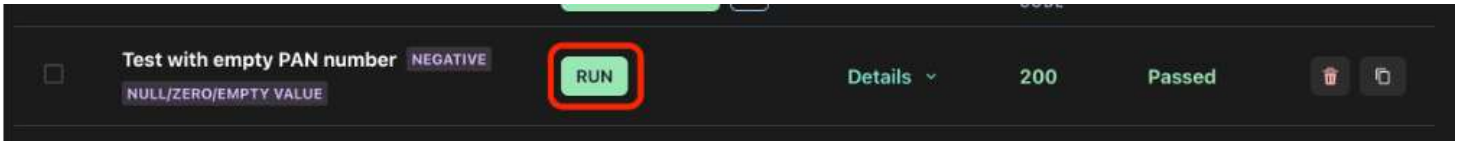
8. **Add Tests using AI:** You can use this to add tests using natural language prompts. Specify what you wish to test in plain English and let KushoAI come up with API payload and assertions.
9. **Pre Run Script:** Use this to run custom logic in JavaScript before starting test suite run. This can be used for automating stuff like fetching a fresh dynamic authentication token from an auth server and setting it in a variable before every run of a test suite.

Running a Single Test:

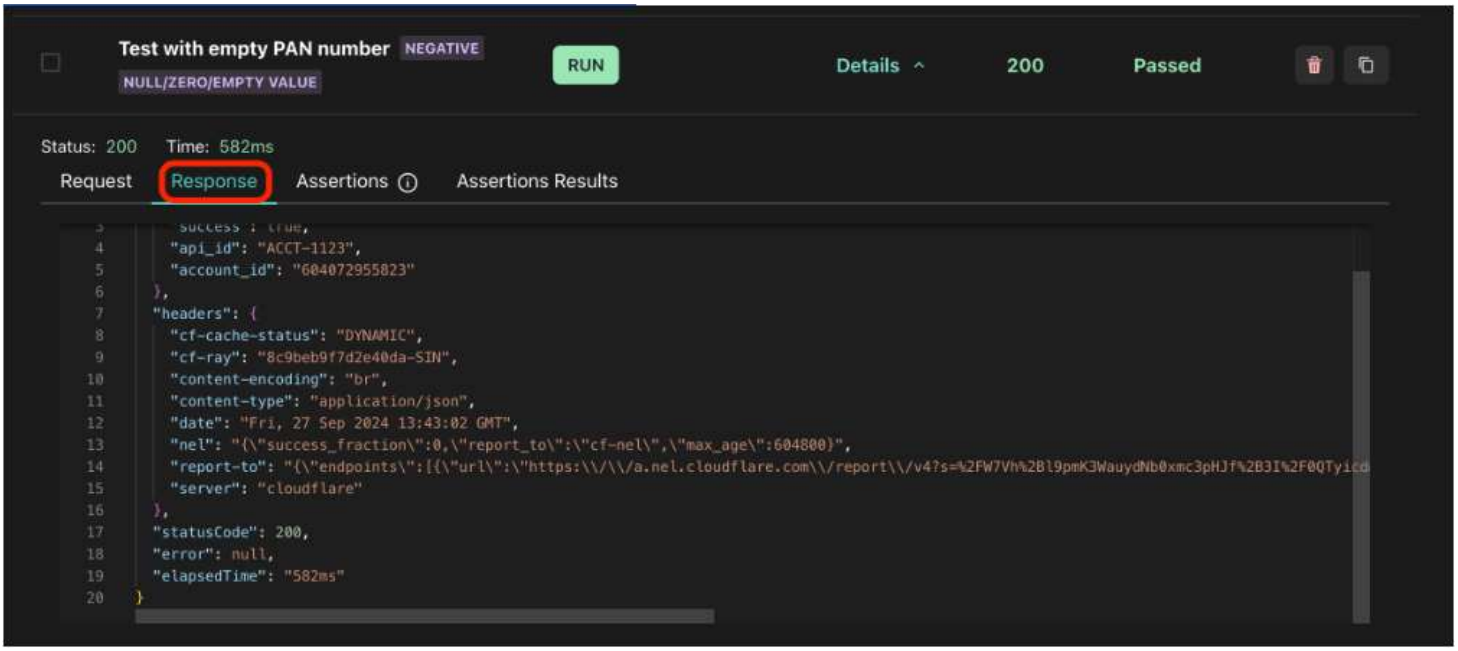
This is the simplest way to run tests on KushoAI. Running like this is useful when you're fixing a specific issue and want to check if it's working as expected by running a particular test.

Here's how you can run a single test and check the results:

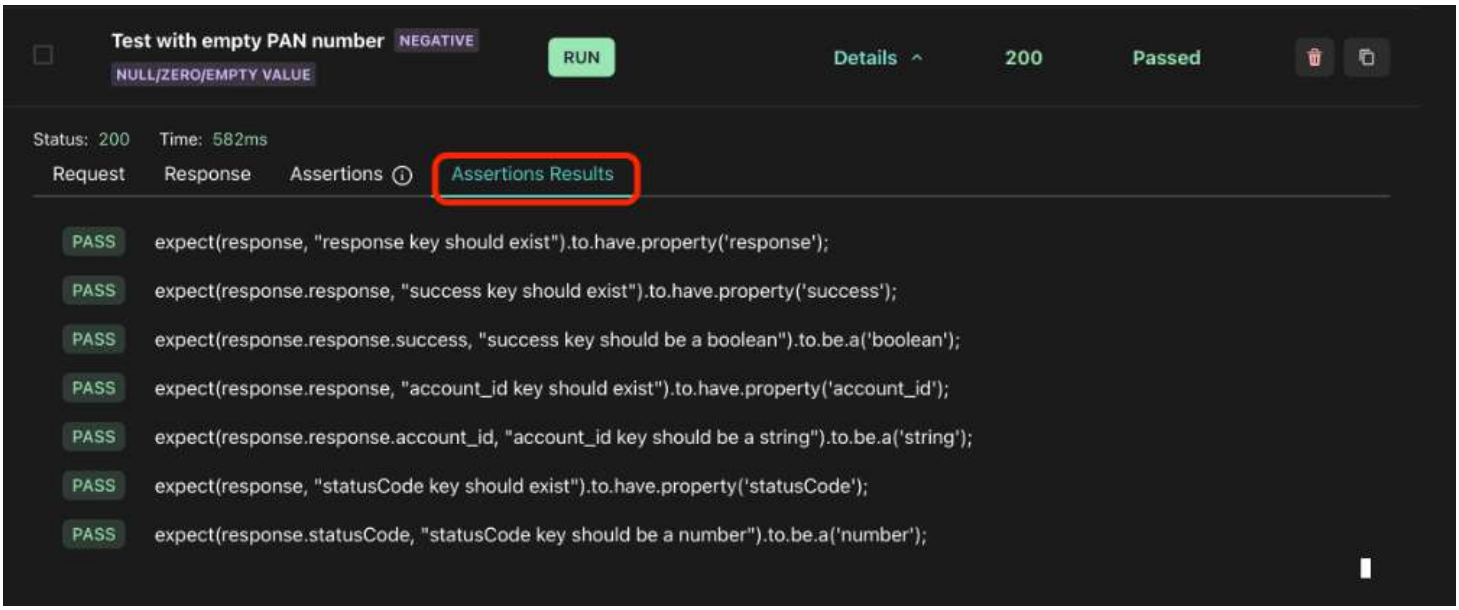
1. Go to the Tests table on Test Suite Details page
2. Click on the "Run" button to execute a test



3. Once the running is completed, you'll see the response in "Response" tab of Test Details section (which can be opened by clicking on "Details" button)



4. "Assertions Results" tab will contain details about which assertions have passed/failed if you have defined assertions



5. Status code and status of assertions is displayed under "Status" and "Assertions" columns respectively.

6. Test counts will be updated once the execution is done. Note that the pass/fail count will change only if you have defined assertions.

Running All Tests:

If you want to run all tests in a test suite in one go, you can use the "Run All" option. Running tests like this is useful to verify if all functionality for an API is working as expected and getting a high level report.

Here's how to use the "Run All" option:

1. Click on "Run All" button located in the column header of Tests table to run all tests in one go

2. By default, tests are run one after another with a 1000ms delay in between runs. If you wish to change this delay and run tests parallely, you can change the configuration using Run settings. Depending on your backend's rate limit and resources, you can tweak the wait time and no. of tests executed in parallel options to make sure your don't end up getting rate limited or crash the backend.
3. Once the running is completed, you'll see the response in "Response" tab of Test Details section (which can be opened by clicking on "Details" button)
4. "Assertions Results" tab will contain details about which assertions have passed/failed if you have defined assertions

5. Status code and status of assertions is displayed under "Status" and "Assertions" columns respectively.
6. Test counts will get updated once the execution is done. Note that pass/fail count will change only if you have defined assertions.
7. You can use the "Generate Report" button to get a high level report of your test run.

Assertions:

Assertions allow you to define expected behaviour of a test. Using assertions, you can automate validation of API response every time a test runs and get a pass/fail status for the test. This works very similar to how asserts work in unit testing. We expect the assertions to be in JS using the Chai.js expect syntax. If you've already used Chai.js or have some experience with unit testing, this should be straightforward for you.

For a bank account creation API, this is the expected response:

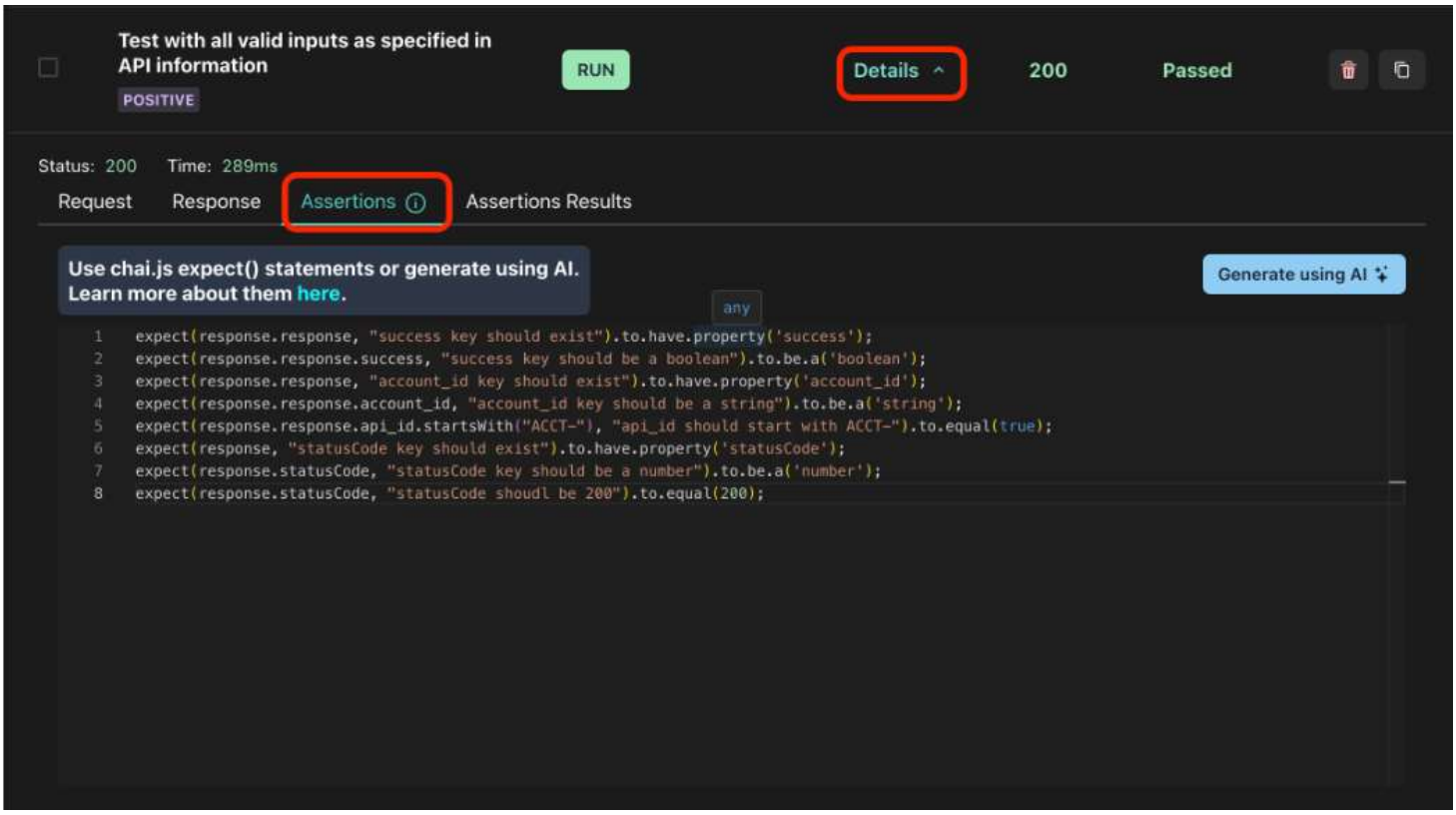

```
{
  "response": {
    "success": true,
    "api_id": "ACCT-1123",
    "account_id": "119471389591"
  },
  "headers": {
    "content-length": "81",
    "content-type": "application/json",
    "date": "Mon, 30 Sep 2024 04:28:36 GMT",
    "server": "Werkzeug/2.0.1 Python/3.9.6"
  },
  "statusCode": 200,
  "error": null,
  "elapsedTime": "290ms"
}
```


Assertions for this expected response will look something like this:

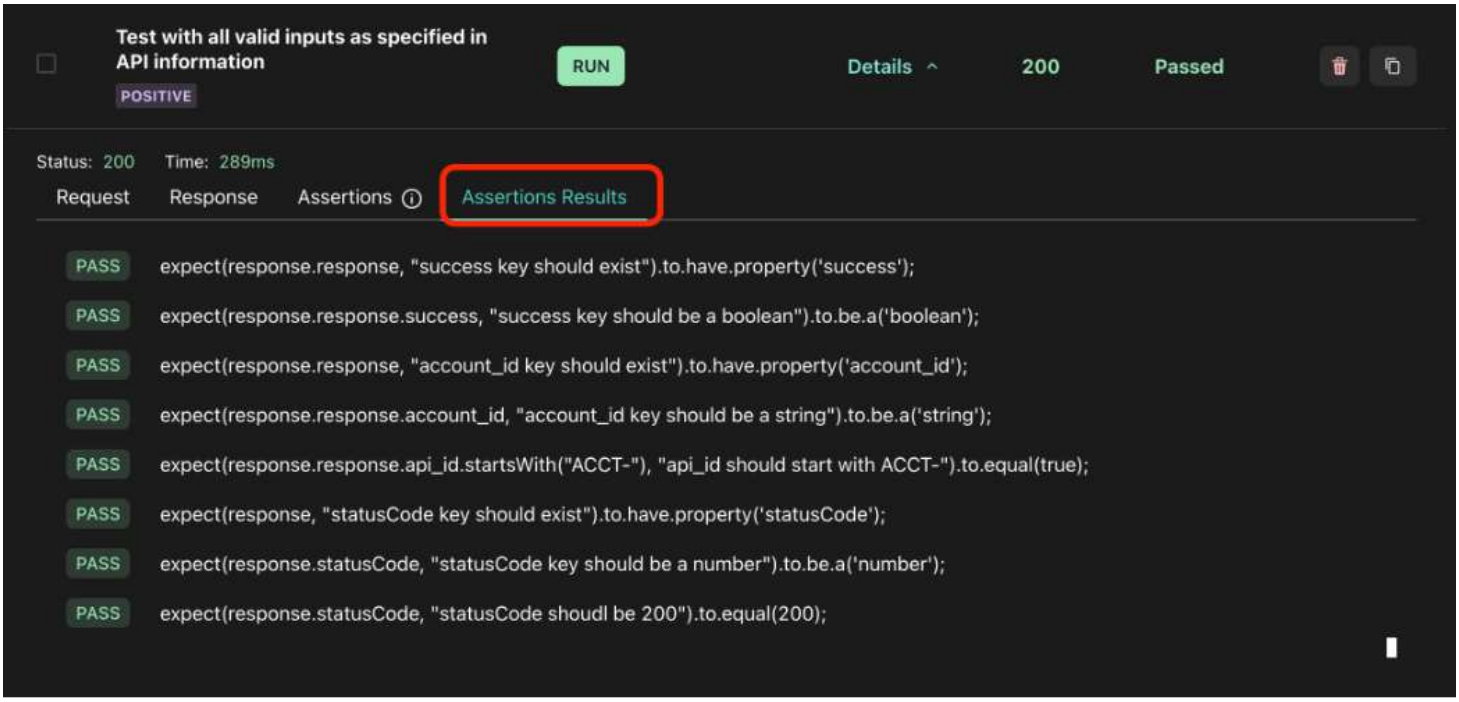
```
expect(response.response, "success key should exist").to.have.property('success');  // check if field is present
expect(response.response.success, "success key should be a boolean").to.be.a('boolean');  // check datatype
expect(response.response, "account_id key should exist").to.have.property('account_id');
expect(response.response.account_id, "account_id key should be a string").to.be.a('string');
expect(response.response.api_id.startsWith("ACCT-"), "api_id should start with ACCT-").to.equal(true);  // check i
expect(response, "statusCode key should exist").to.have.property('statusCode');
expect(response.statusCode, "statusCode key should be a number").to.be.a('number');
expect(response.statusCode, "statusCode shoudl be 200").to.equal(200);  // check if field value is correct
```

Note that the top-level response object is called response. This object contains the API response (under response field) along with other additional fields like headers, statusCode, elapsedTime (response time of the API), etc. that might be useful for assertions. Fields inside the response object can be accessed like response.<field-name>.

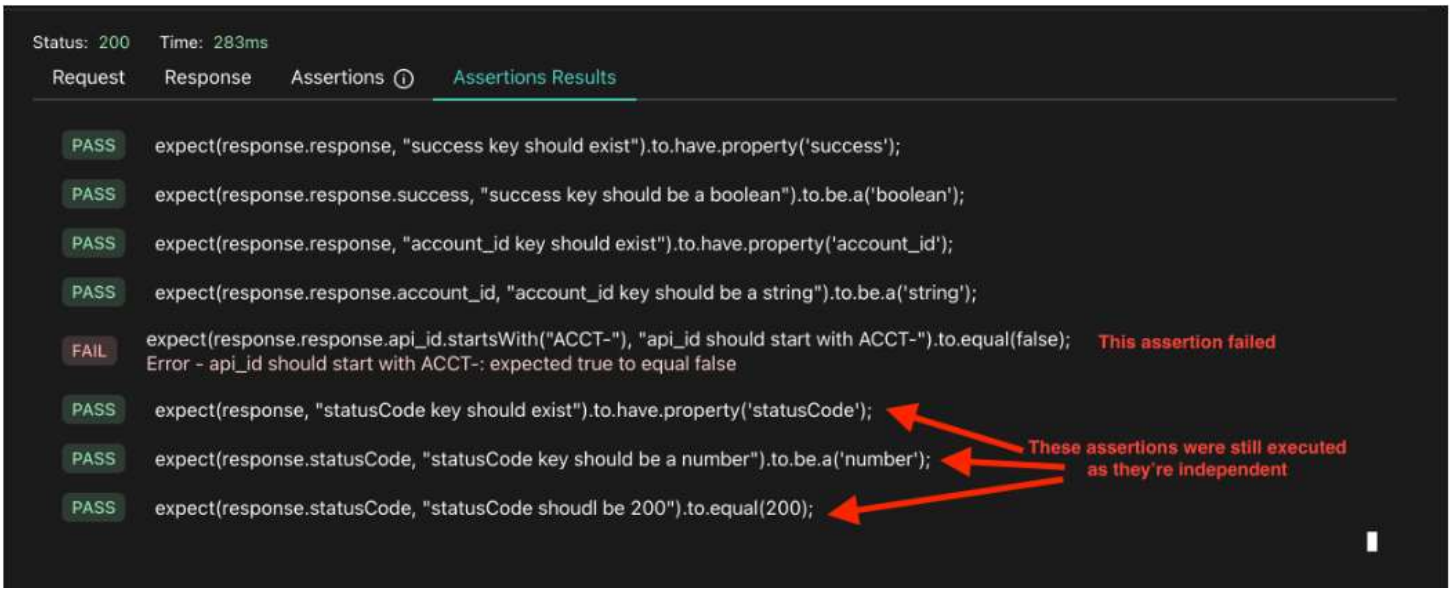
Assertions are executed after running a test. You can see them under the "Assertions" tab in Test details:



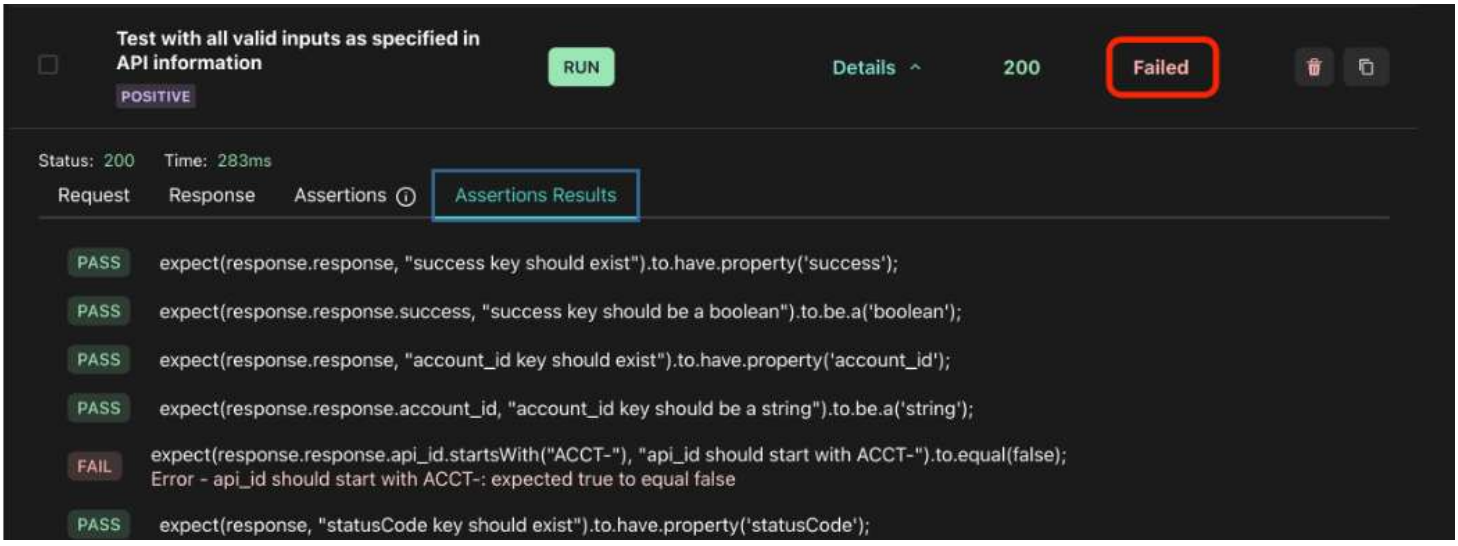
The results are shown under the "Assertion Results" tab. Here, you'll see the status for every assertion you've added.



All assertions are executed independently of each other. So even if 1 assertion fails, all assertions after it will still get executed.



We mark a test as failed even if a single assertion fails. This status is shown under the "Assertions" column of Test table. You will see a blank or "N/A" here, if you've not written assertions for a test.



The pass/fail count under Test counts is calculated using a test's pass/fail status. Note that this count changes only if your tests have assertions.

Test Reports:

The Test Report feature in Kusho provides a comprehensive view of API test execution results. The report presents test outcomes in a clear tabular format, making it easy to track and analyze test suite performance.

Report Structure:

Summary Section:

The report header includes key metrics:

- Total number of tests
- Number of executed tests
- Number of passed tests
- Number of failed tests
- Total execution time (in milliseconds)
- Execution timestamp (UTC)
- Executor's email

Test Suite Details:

Each test suite is presented with:

- Suite name
- Complete test case listing
- Individual test results
- Status codes
- Failed assertions (if any)

Report Components:

Test Case Information

Each test case includes:

- No. (Sequential number)
- UUID (Unique identifier)
- Test Case description
- Result (PASS/FAIL)
- Status Code
- Failed Assertions (N/A if passed)

Result Categories:

- PASS - Test executed successfully

- FAIL - Test execution failed
- Status Code - HTTP response code (e.g., 401)
- Failed Assertions - Details of any failed test conditions

Features:

- Comprehensive test execution summary
- Detailed individual test results
- Clear pass/fail indicators
- HTTP status code tracking
- Failed assertion tracking
- Unique test case identification

Use Cases:

- API testing validation
- Quality assurance documentation
- Test coverage reporting
- Regression testing verification
- Compliance documentation

Report Structure:

KUSHO

# Total	# Executed	# Passed	# Failed	Time (ms)	Executed At	Executed By
17	17	17	0	5842	2025-03-07 07:20:59 UTC	akash@[redacted]com

Test suite - Teams - Team List

No.	UUID	Test Case	Result	Status Code	Failed Assertions
1	6d9c8308-0964-4872-8b1b-27d08f8b6aa3	Test with array value for query parameter 'foo'	PASS	200	N/A
2	b71010c3-cd9b-42f9-bdd3-7d1851bbcfc1	Test with boolean value (true) for query parameter 'foo'	PASS	200	N/A
3	7e17f8e0-bf33-48d4-81f3-914dad7ebddf	Test with empty value for query parameter 'foo'	PASS	200	N/A
4	3d494a2b-6547-4604-94f7-c0a1c4ca38d9	Test with special characters for query parameter 'foo'	PASS	200	N/A
5	a7cb0d64-bcc2-4fe2-b735-283013fc6bcf	Test with integer value for query parameter 'foo'	PASS	200	N/A
6	1600fa90-0d42-4850-ae05-7d77dbddb564	Test with leading and trailing spaces for query_params.foo field	PASS	200	N/A
7	73b15701-bb28-4eb4-b592-a1f4f817e385	Test with integer value as a string for query_params.foo field	PASS	200	N/A
8	e7ad4fe4-45e3-476a-aa0b-e603cfa961bf	Test with very long string for query_params.foo field	PASS	200	N/A
9	710f79da-9e0f-4e24-bb0b-1e19f8fc81e1	Test with non-English characters string for query_params.foo field	PASS	200	N/A
10	ab935a9f-8d68-4725-9ce6-cb85d0ba0e70	Test with all lowercase string for query_params.foo field	PASS	200	N/A
11	ed39e99c-115a-426f-a936-3b546c357883	Test with all uppercase string for query_params.foo field	PASS	200	N/A
12	983e648b-826b-4755-a93f-e43e755dfb6d	Test with null value for query_params.foo field	PASS	200	N/A
13	71637a31-8822-44ce-9c8f-f5b8279d7669	Test with empty string for query_params.foo field	PASS	200	N/A
14	ec6276b5-7372-4812-a8f7-b498b621cb9f	Test with valid input for query_params.foo field	PASS	200	N/A
15	4154627f-36cc-4e82-8bc8-173375aa4e5e	Test with given field missing - query_params.foo	PASS	200	N/A

Conclusion:

KushoAI introduces a significant shift in API testing by automating the generation of comprehensive and high-coverage test suites using AI-driven logic. Its ability to parse OpenAPI specifications, Postman Collections, and cURL commands allows it to quickly identify input parameters, expected outputs, edge cases, and potential failure scenarios. The tool streamlines test creation and validation while supporting integration into CI/CD pipelines, enabling continuous and scalable API quality assurance. For engineering teams seeking to enhance test reliability, maintainability, and speed, KushoAI provides a robust and intelligent solution that aligns well with modern DevOps practices.