Test Strategy

1. Web services/ API Protocols

- **REST** - REST stands for Representational State Transfer
- **SOAP API** - Stands for Simple Object Access Protocol.

2. Approach for API testing

- Inspecting API Spec/Doc
- Designing functional/service level test cases
- Designing integration tests
- Perform security, performance and load testing once the API is stabilized

3. Types of API Testing

- Unit Testing
- Functional Testing
- Load Testing
- Security Testing
- Interoperability Testing
- WS compliance Testing
- Penetration Testing

4. API test actions

- **Verify correct HTTP status code -** For example, creating a resource should return 201 CREATED and unpermitted requests should return 403 FORBIDDEN, etc.
- **Verify response payload -** Check valid JSON body and correct field names, types, and values including in error responses.
- **Verify json response against DB values -** Values in DB should match the respective API response value on comparison.

- **Verify response headers -** HTTP server headers have implications on both security and performance.
- **Verify correct application state -** This is optional and applies mainly to manual testing, or when a UI or another interface can be easily inspected.
- **Verify basic performance sanity -** If an operation was completed successfully but took an unreasonable amount of time, the test fails.

5. Test scenario categories

- Basic positive tests (happy paths)
- Extended positive testing with optional parameters
- Negative testing with valid input
- Negative testing with invalid input
- Destructive testing
- Security, authorization, and permission tests

6. Test flows

- Testing request in isolation (Executing Individual API's)
- Multi-step workflow with several requests(Executing API's at component level)
- End-to-End Testing (Combined API and web UI test)

7. Tools for API Automation

- RestAssured
- SOUP UI
- Katalon studio
- Postman
- Jmeter

- CloudQA TruAPI etc.,

8. An API example and a test matrix

#	Test Scenario Category	Test Action Category	Test Action Description
1	**Basic positive tests (happy paths)**		
	Execute API call with **valid required parameters**	Validate status code:	1. All requests should return 2XX HTTP status code 2. Returned status code is according to spec: – 200 OK for GET requests – 201 for POST or PUT requests creating a new resource – 200, 202, or 204 for a DELETE operation and so on
		Validate payload:	1. Response is a well-formed JSON object 2. Response structure is according to data model (schema validation: field **names** and field **types** are as expected, including nested objects; field **values** are as expected; non-nullable fields are not null, etc.)
		Validate state:	1. For GET requests, verify there is NO STATE CHANGE in the system. 2. For POST, DELETE, PATCH, PUT operations – Ensure action has been performed correctly in the system by: – Performing appropriate GET request and inspecting response
		Validate headers:	Verify that HTTP headers are as expected, including content-type, connection, cache-control, expires, access-control-allow-origin, keep-alive, HSTS and other standard header fields – according to spec. Verify that information is NOT leaked via headers.

		Performance sanity:	Response is received in a timely manner (within reasonable expected time)
2	**Positive + optional parameters**		
	Execute API call with **valid required parameters AND valid optional** parameters Run same tests as in #1, this time including the endpoint's optional parameters (e.g., filter, sort, limit, skip, etc.)		
		Validate status code:	As in #1
		Validate payload:	Verify response structure and content as in #1. In addition, check the following parameters: – filter: ensure the response is filtered on the specified value. **– sort:** specify field on which to sort, test ascending and descending options. Ensure the response is sorted according to selected field and sort direction. **– skip:** ensure the specified number of results from the start of the dataset is skipped – limit: ensure dataset size is bounded by specified limit. **– limit + skip:** Test pagination Check combinations of all optional fields (fields + sort + limit + skip) and verify expected response.
		Validate state:	As in #1
		Validate headers:	As in #1
		Performance sanity:	As in #1
3	**Negative testing – valid input**		
	Execute API calls with **valid input** that attempts illegal		

	operations. i.e., – Attempting to create a resource with a name that already exists (e.g., creating the city with zipcode already exists) – Attempting to delete a resource that doesn't exist (e.g., deleting the city with no such city ID) – Attempting to update a resource with illegal valid data (e.g., rename a city to an existing city id) – Attempting illegal operation (e.g., delete a city configuration without permission.) And so forth.		
		Validate status code:	. Verify that an erroneous HTTP status code is sent (NOT 2XX) 2. Verify that the HTTP status code is in accordance with error case as defined in spec
		Validate payload:	1. Verify that error response is received 2. Verify that error format is according to spec. e.g., error is a valid JSON object or a plain string (as defined in spec) 3. Verify that there is a clear, descriptive error message/description field 4. Verify error description is correct for this error case and in accordance with spec
		Validate headers:	As in #1
		Performance sanity:	Ensure error is received in a timely manner (within reasonable expected time)
4	**Negative testing – invalid input**		
	Execute API calls with invalid input, e.g.:		

	– Missing or invalid authorization token – Missing required parameters – Invalid value for endpoint parameters, e.g.: – Invalid APP Id in path or query parameters – Payload with invalid model (violates schema) – Payload with incomplete model (missing fields or required nested entities) – Invalid values in nested entity fields – Invalid values in HTTP headers – Unsupported methods for endpoints And so on.		
		Validate status code:	As in #1
		Validate payload:	As in #1
		Validate headers:	As in #1
		Performance sanity:	As in #1
5	**Destructive testing**		
	Intentionally attempt to fail the API to check its robustness: Malformed content in request Wrong content-type in payload Content with wrong structure Overflow parameter values. E.g.: – Attempt to create a user configuration with a title longer		

	than 200 characters – Attempt to GET a user with invalid city ID which is 1000 characters long – Overflow payload – huge JSON in request body Boundary value testing Empty payloads Empty sub-objects in payload Illegal characters in parameters or payload Using incorrect HTTP headers (e.g. Content-Type) Small concurrency tests – concurrent API calls that write to the same resources (DELETE + PATCH, etc.) Other exploratory testing		
		Validate status code:	As in #3. API should fail gracefully.
		Validate payload: Validate headers:	As in #3. API should fail gracefully. As in #3. API should fail gracefully.
		Performance sanity:	As in #3. API should fail gracefully.

9. Measures and Metrics

Test Preparation

- Number of Test Scenarios v. Number of Test Cases
- Number of Test Cases Planned v. Ready for Execution
- Total time spent on Preparation v. Planned time

Test Execution and Progress

- Number of Tests Cases Executed v. Test Cases Planned
- Number of Test Cases Passed, Failed and Blocked
- Total Number of Test Cases Passed by Test Item / Test Requirements
- Total Time Spent on Execution vs Planned Time

Bug Analysis

- Total Number of Bugs Raised and Closed per Test Run
- Total Number of Bugs Closed v. Total Number of Bugs Re-Opened
- Bug Distribution Totals by Severity per Test Run
- Bug Distribution Totals by Test Item by Severity per Test Run