Basics / HTTP Concepts

1. Common HTTP Methods in API?

The most common HTTP methods in API are GET, POST, PUT, PATCH, and DELETE. They are used to read, create, update (full/partial), and delete resources respectively.

2. Different common status codes in API? (2xx, 3xx, 4xx, 5xx)

2xx (Success) – 200 OK, 201 Created, 204 No Content

3xx (Redirection) – 301 Moved Permanently, 302 Found, 304 Not Modified

4xx (Client Error) – 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found

5xx (Server Error) – 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable

3. Difference between PUT and PATCH?

PUT is used to update the entire resource, while PATCH is used to update only specific fields of a resource.4. Difference between Query Params and Path Params?

4.Query Params vs Path Params (Interview-style short answer):

Path parameters are used to identify a specific resource, while query parameters are used to filter, sort, or modify the response.

5.And can you use PUT before POST? Which comes first?

No, normally you **cannot use PUT before POST**. **POST comes first** to create the resource, and then PUT is used to update it.

6. What does 409 mean?

409 Conflict means the request could not be completed because of a conflict with the current state of the resource.

Authorization & Security

1. Authorization in automation testing?

Authorization in automation testing is the process of validating whether a user has the correct permissions and access rights to perform specific actions on an application or API.

2. Imagine you are testing a login page, how would you ensure that it is secure against all invalid data (without using automation)?

To ensure a login page is secure against invalid data, I would perform negative testing by trying invalid combinations like wrong username/password, empty fields,

special characters, SQL injection, and very long inputs, and verify proper error messages without exposing system details.

3. "Equivalence Partitioning" – How does this test design technique help in testing the login page?

Equivalence Partitioning helps in testing a login page by dividing input data into valid and invalid groups, so we test only one value from each group instead of testing all possible inputs.

4. About Maximum and minimum input values, to which field were you referring? Username or password?

Maximum and minimum input values apply to both username and password, because both fields have length and format constraints that must be validated.Negative Scenarios & Error Handling.

5. How do you handle negative scenarios in API?

I handle negative scenarios in API testing by sending invalid, missing, or unexpected inputs, including invalid or expired tokens, and then verify that the API returns the correct error status codes (like 400, 401, 403, 404, 409) along with meaningful and secure error messages.

6. And you want to put those negative scenario data in a database using API. So what steps are you going to do?

To store negative scenario data in a database using an API, I first design the negative test cases, prepare invalid inputs, call the API with those inputs, capture the responses, and then validate that the API correctly stores the error details in the database.

7. If you are running the POST command and it is failing, what might be the issues?

If a POST request is failing, the common issues could be invalid payload, missing mandatory fields, wrong content-type, authentication/authorization issues, or server-side validation errors.

8. In a front-end, you see HTTP 200 OK, but it retrieves zero records from your DB. How will you resolve this issue?

If the front-end shows **200 OK but returns zero records**, it means the request was successful, but the **data retrieval logic or filters are incorrect**. I would debug by checking request parameters, API logic, and database queries.

Dynamic Values & Schema Validation

---

1. If you are working with dynamic values in API, what steps will you follow?

When working with dynamic values in API, I first **capture the value from a previous response**, store it as a variable, and then **pass it to the next request** for validation or chaining.

2. About schema validations? What about nested JSON objects?

Schema validation ensures that the API response follows the expected **JSON structure, data types, and mandatory fields**, and it also works for **nested JSON objects** by validating each level of the hierarchy.

3. If you are performing API testing, what information do you get from the developer?

From the developer, I get the **API contract/specification**, including endpoints, HTTP methods, request/response formats, headers, authentication details, and expected status codes.

Testing Approach & Scenarios

1. If you have created 10–20 test cases, how many would you automate vs manual?

Out of 10–20 test cases, I would automate around **70–80%**, mainly the stable and repetitive scenarios, and keep the remaining **20–30%** manual for unstable, exploratory, or one-time cases based on stability and business priority.

2. What kind of tests do you perform in REST API? Provide examples and explain your approach.

In REST API testing, I perform **functional, negative, validation, security, performance, and integration tests** to ensure the API works correctly, handles errors properly, and meets business requirements.

6. How do you design a test case or a test suite for API or any web applications?

I design test cases or a test suite by first understanding the **requirements and business logic**, then identifying test scenarios, preparing test data, and finally covering **positive, negative, and edge cases**.

7. How did you perform integration testing in your API automation framework?

I performed integration testing by **chaining multiple related APIs**, passing dynamic values from one response to the next request, and validating the **end-to-end data flow** across systems.

8. Have you extended support for production validation? Give examples.

Yes, I have provided **production validation support** by performing **smoke checks and sanity testing** after deployments to ensure critical APIs and functionalities are working as expected.

Challenges & Reporting

1. What are the challenges you faced in API testing?

The main challenges I faced in API testing are unclear documentation, complex authentication, test data issues, unstable environments, handling dynamic responses, and dependency failures between microservices.

10. If there is a conflict with a developer about a defect, how would you approach it?

I approach conflicts collaboratively by reproducing the issue, sharing clear evidence, referring to requirements, discussing calmly with the developer, and escalating only if needed.

11. Suppose you need to report a bug at the last moment, what should your defect report include?

A last-minute defect report should include clear steps to reproduce, environment, expected vs actual result, logs/screenshots, test data, and severity/priority so it can be fixed quickly.

Broken Links

1. How do you find broken links (Dead Links) in API?

I find broken links in APIs by sending requests to all endpoints and validating the HTTP status codes. If an endpoint returns 4xx or 5xx errors like 404 or 410 instead of a successful 200/201 response, I consider it a dead link. I usually automate this using tools like Postman, curl, or test frameworks to loop through endpoints and report failures.

2. How do you find broken links (Dead Links) in UI?

I find broken links in UI by extracting all hyperlinks from the page and validating their HTTP responses. If a link returns 4xx or 5xx status codes like 404 or 500 instead of 200, I mark it as a dead link. This can be done manually using browser DevTools or automatically using automation tools like Selenium/Playwright by sending HTTP requests to each link.

Manual Testing & QA Scenarios

1. How much experience do you have in understanding requirements and deriving test scenarios?

I have around 6 years of overall experience in testing, with about 4 years in automation and strong hands-on experience in understanding business requirements, analyzing user stories, and deriving detailed test scenarios and test cases for both API and UI testing. I regularly collaborate with product owners and

developers to clarify requirements and ensure proper coverage of functional, negative, and edge cases.

2. Can you explain the difference between functional and non-functional testing?

Functional testing focuses on verifying what the system does, meaning it checks whether the application's features and business logic work as per requirements, such as validating APIs, UI flows, and user actions.
Non-functional testing focuses on how the system performs, meaning it checks aspects like performance, load, security, usability, reliability, and scalability to ensure the system meets quality standards beyond just functionality.

3. Major difference between smoke and sanity testing?

Smoke testing is a **high-level test** done on a new build to verify that the critical functionalities are working and the build is stable for further testing.
Sanity testing is a **focused test** done after minor changes or bug fixes to verify that a specific functionality works as expected without doing full regression.

4. Imagine a situation where the requirement document is missing or incomplete. How would you approach it?

f the requirement document is missing or incomplete, I would first discuss with the product owner, business analyst, or developers to understand the expected behavior. I would also analyze existing application functionality, refer to similar features, and check API contracts or user stories. Based on this, I would derive assumptions, validate them with stakeholders, and then prepare test scenarios to ensure proper coverage.

5. Did you maintain clarification logs? What was their purpose and how would you approach it?

Yes, I have maintained clarification logs. Their purpose is to track open questions, assumptions, and ambiguities in requirements. I use them to raise queries with stakeholders, document decisions, and ensure everyone has the same understanding. This helps avoid misunderstandings, improves requirement clarity, and acts as a reference during test case design and execution.

CI/CD Questions

---

1. What is a CI/CD pipeline?

A CI/CD pipeline is an automated process that allows teams to continuously integrate code changes, run automated tests, and continuously deploy the application to different environments. It helps detect defects early, ensure faster releases, and maintain software quality through automated build, test, and deployment stages.

2. Can you explain Jenkins and its role in automation?

Jenkins is an open-source automation server used to implement CI/CD pipelines. It helps automate tasks like building code, running automated tests, and deploying applications. In automation, Jenkins is used to trigger test suites on every code change, schedule test executions, generate reports, and integrate with tools like Git, Maven, Pytest, and Playwright to ensure continuous testing and faster feedback.

3. Imagine you ran execution using a YAML file and it is successful, but the automated report is not generated. What will you do?

First, I would check the pipeline logs to ensure the report generation step actually ran and did not fail silently. Then I would verify the report configuration in the YAML file, such as report path, plugin settings, and permissions. I would also check if the reporting tool is properly installed and integrated. If needed, I would rerun the job with debug logs, validate the workspace for generated files, and coordinate with the DevOps team if it's a pipeline configuration issue.

Pytest

---

1. What is the `requests` library in Python?

The requests library in Python is a simple and powerful HTTP client used to send API requests like GET, POST, PUT, and DELETE. It allows us to interact with web services easily by handling headers, authentication, parameters, and responses, and is widely used for API testing and automation.

2. How do you make a GET request using `requests`?

I make a GET request using the requests library by calling the get() method with the API URL and optional headers or parameters, then I validate the response using status code and response body.

3. How can you send data using POST request?

I send data in a POST request using the requests.post() method by passing the payload in the request body, usually as JSON using the json parameter or as form data using the data parameter, along with required headers like Content-Type.

4. How do you handle headers in a request?

I handle headers in a request by passing them as a dictionary using the headers parameter in the requests methods. This is used to send information like Content-Type, Authorization tokens, and custom headers along with the API request.

5. What if the request fails or times out?

If a request fails or times out, I handle it by setting a timeout value, using exception handling to catch errors, and adding assertions to report meaningful failure

messages. I also log the request and response details, retry if required, and analyze whether the issue is due to environment, network, or application problems.

6. How do you send JSON in a POST request?

I send JSON in a POST request by using the json parameter in the requests.post() method and setting the Content-Type header to application/json, which automatically serializes the dictionary into JSON format.

7. How to read response content?

I read the response content using attributes like response.text for raw text, response.json() for JSON responses, and response.content for binary data, depending on the response type.

8. How to send query parameters?

I send query parameters by passing them as a dictionary using the params argument in the request method, which automatically appends them to the URL.

9. Can you upload files using `requests`?

Yes, I can upload files using the requests library by using the files parameter in a POST request, where the file is sent as multipart form data.

10. Real-world example use-case?

In a real-world scenario, I use the requests library for API automation, such as testing a login API by sending credentials in a POST request, validating the response token, then using that token in headers for subsequent GET or POST requests to verify authorized access to other endpoints.

11. How does "yield" works in pytest frameworks?

In pytest, yield is mainly used in fixtures for setup and teardown. The code before yield runs as setup, and the code after yield runs as teardown. It helps manage resources like opening and closing connections, starting and stopping services, or creating and cleaning up test data.

AI / ML Related Testing

---

1. In LLM, what is your role, what business problem were you solving, and how did you tackle it?

My role was to work as an automation tester focusing on integrating LLM with API testing. The business problem was reducing manual effort in test case creation and improving test coverage for complex APIs. I tackled it by using LLM to generate test scenarios from requirements and Swagger specs, converting them into pytest-based API tests, validating responses automatically, and integrating the solution into the CI/CD pipeline for continuous testing.

2. What kind of testing did you perform using LLM? How did you validate test cases and scripts? Provide examples.

I mainly used LLM for functional API testing and test case generation. I used it to generate positive, negative, and edge-case scenarios from requirements, Swagger files, and sample payloads. To validate the test cases, I reviewed them manually for business correctness, mapped them with acceptance criteria, and executed them using pytest. I validated scripts by checking response status codes, schema validation, and key field assertions, and compared results with manual executions in Postman to ensure accuracy.

3. Give examples of AI features you tested, input you provided, and expected output.

I tested features like automated test case generation, intelligent data validation, and response analysis using LLM. For example, I provided API requirements and Swagger specs as input, and expected the model to generate functional and negative test scenarios. I also tested prompt-based validation where I passed API responses as input and expected the LLM to identify missing fields, incorrect values, or business rule violations. The expected output was accurate test cases, meaningful assertions, and correct defect identification aligned with requirements.

4. What exactly did you work on with AI/ML features using DeepEval automation framework?

I worked on validating LLM-based features using the DeepEval framework, mainly focusing on evaluating prompt responses, accuracy, relevance, and consistency of outputs. I used DeepEval to automate validation of AI-generated test cases and responses by defining evaluation metrics, passing different prompts as inputs, and asserting the model's outputs against expected behavior such as correctness, completeness, and hallucination checks.

5. How did you approach AI/ML for regression testing, especially via APIs?

I approached AI/ML regression testing by creating automated API test suites that send prompt-based requests to LLM endpoints and validate responses using predefined metrics. I compared current outputs with baseline responses, checked for response drift, accuracy, and consistency, and used tools like pytest and DeepEval to detect regressions in model behavior after new releases or prompt changes.

6. How do you handle anomaly detection in API responses?

I handle anomaly detection by defining baseline behavior for API responses and validating current responses against it. I check for unexpected status codes, missing or extra fields, schema mismatches, abnormal values, and response time deviations. For AI/ML APIs, I also compare outputs with historical results to detect response drift, inconsistencies, or hallucinations, and flag any deviations using automated assertions and evaluation metrics.

7. API response schema validation using AI/ML approach?

I perform API response schema validation using an AI/ML approach by defining the expected schema as a baseline and using LLM or tools like DeepEval to validate responses against it. The model checks for missing or extra fields, incorrect data types, and structural mismatches. For dynamic responses, I use AI to intelligently compare patterns and flag anomalies that traditional schema validation might miss, such as semantic errors or inconsistent field values.

8. What LLM tools and libraries did you use for API automation with Pytest?

I used tools and libraries like OpenAI/LLM APIs, LangChain for prompt orchestration, DeepEval for evaluation and validation, and pytest as the core automation framework. I also used standard libraries like requests for API calls, pydantic or JSON schema for response validation, and integrated everything with CI/CD pipelines like Jenkins for continuous execution and reporting.

9. Purpose of hallucination check, and how do you tackle it?

The purpose of hallucination check is to ensure that the LLM does not generate incorrect, fabricated, or irrelevant information. I tackle it by validating AI responses against trusted sources like API specs, business rules, and ground-truth data. I also use tools like DeepEval with predefined metrics, baseline comparisons, and prompt constraints to detect inconsistencies and flag any outputs that deviate from expected factual behavior.

10. Explain LLM-EB testing. Why is it needed in API?

LLM-EB (Evaluation-Based) testing is a testing approach where LLM outputs are validated using defined evaluation metrics instead of only traditional assertions. It is needed in APIs because AI responses are often non-deterministic and cannot always be validated using fixed expected values. LLM-EB testing helps measure correctness, relevance, consistency, and hallucination, ensuring the AI-powered API behaves reliably across different inputs and releases.

11. Explain NUMPY, Matplotlib, PANDAS, HPPTX, Requests libraries and their usage in API automation.

NumPy is used for numerical operations and data manipulation, especially for handling large datasets from API responses.
Pandas is used for reading, cleaning, and analyzing structured data like CSV or JSON responses in API testing.
Requests is used to send HTTP requests such as GET, POST, PUT, and DELETE to test APIs.
Matplotlib is used to visualize API test results, such as plotting response times, pass/fail trends, and performance metrics.
Python-PPTX is used to automatically generate test reports and dashboards in PowerPoint format from API execution results for stakeholders.

12. When you have prediction models, explain what type of model you used for predictions.

I mainly worked with supervised learning models for predictions, where the model is trained on labeled historical data to predict future outcomes. Depending on the use case, I used regression models for numerical predictions and classification models for categorical outcomes. For more complex scenarios, I also used pre-trained LLM-based models and fine-tuned them to predict trends, classify inputs, or generate intelligent recommendations based on patterns in the data.

SQL Questions

1. Write a SQL query to find the first transaction done by all users for all days. (Example: Today $10, $120, Tomorrow $200, etc.)

I use ROW_NUMBER() with PARTITION BY user_id, transaction_date and order by transaction time. This assigns a rank to each transaction per user per day, and selecting rn = 1 gives the first transaction done by each user for each day.