

REST ASSURED

API Test Automation Interview Q&A

A Complete Preparation Guide for QA Automation Engineer Roles

Covers Requests · Validation · Auth · Schema Testing · CI Integration

-
- ✓ Java-based Rest Assured questions written from a QA automation engineer's perspective
 - ✓ Beginner to advanced difficulty, organized by topic
 - ✓ Real code snippets you can reuse in interviews and on the job
 - ✓ Includes negative testing and BDD (Cucumber) integration scenarios
 - ✓ A quick-reference cheat sheet at the end

Prepared as a study reference for QA Automation Engineer interviews. Practice explaining each answer in your own words — interviewers value clear reasoning over memorized definitions.

Table of Contents

Section A	Rest Assured Fundamentals	5 Qs
Section B	Authentication, Headers & Request Configuration	5 Qs
Section C	Validation Strategies & Response Handling	5 Qs
Section D	Framework Design, BDD & CI Integration	5 Qs
Appendix	Quick-Reference Cheat Sheet	

Section A | Rest Assured Fundamentals

Q. What is Rest Assured and why is it commonly used by QA automation engineers for API testing?

Rest Assured is a Java library that provides a fluent, Given-When-Then DSL for sending HTTP requests and validating responses, making API tests read almost like plain English test specifications. It integrates naturally with TestNG/JUnit and Java-based Selenium frameworks, so teams can keep both UI and API automation in a single codebase and CI pipeline.

Q. Explain the Given-When-Then structure in Rest Assured.

given() sets up preconditions like headers, auth, and request body; when() specifies the actual HTTP action (get/post/put/delete) and endpoint; then() defines the validations against the response (status code, body content, headers). This structure mirrors BDD phrasing and keeps tests readable even for non-developers reviewing them.

```
given()
  .baseUrl("https://api.example.com")
  .header("Content-Type", "application/json")
  .when()
    .get("/users/1")
  .then()
    .statusCode(200)
    .body("name", equalTo("John Doe"));
```

Q. How do you send a POST request with a JSON body in Rest Assured?

Set the request body using .body() with either a raw JSON string, a Map, or a POJO (automatically serialized if a JSON serializer like Jackson/Gson is on the classpath), and specify the content type header.

```
Map<String, Object> payload = new HashMap<>();
payload.put("name", "Jane");
payload.put("job", "QA Engineer");

given()
  .contentType(ContentType.JSON)
  .body(payload)
  .when()
    .post("/users")
  .then()
    .statusCode(201);
```

Q. How do you validate the response body using JSON path in Rest Assured?

Use .body() or .path() assertions with JsonPath expressions to drill into nested JSON structures and arrays, combined with Hamcrest matchers for flexible comparisons (equalTo, hasItem, greaterThan, etc.).

```
given().when().get("/users").then()
  .body("data.size()", greaterThan(0))
  .body("data[0].email", containsString("@"))
  .body("data.id", hasItem(2));
```

Q. How would you extract a value from one API response to use in a subsequent request?

Capture the response, extract the needed field via `JsonPath/response.path()`, and pass it into the next request — a common pattern for workflows like create-then-verify or authenticate-then-use-token.

```
Response createResponse = given().contentType(ContentType.JSON)
    .body(payload).post("/users");
String userId = createResponse.jsonPath().getString("id");

given().pathParam("id", userId)
    .when().get("/users/{id}")
    .then().statusCode(200);
```

Section B | Authentication, Headers & Request Configuration

Q. How do you handle Basic Auth and Bearer Token authentication in Rest Assured?

Basic Auth uses `.auth().preemptive().basic(user, pass)`; Bearer tokens are typically passed as a header since Rest Assured has no dedicated bearer-token method by default.

```
// Basic Auth
given().auth().preemptive().basic("user", "pass").when().get("/secure");

// Bearer Token
given().header("Authorization", "Bearer " + token)
.when().get("/secure");
```

Q. How would you design a reusable request specification for a suite of API tests?

Use `RequestSpecBuilder` to define shared configuration (base URI, common headers, content type, auth) once, then reuse the built `RequestSpecification` across all tests, avoiding duplicated setup and making a base-URI/environment change a one-line update.

```
RequestSpecification spec = new RequestSpecBuilder()
.setBaseUri("https://api.example.com")
.setContentType(ContentType.JSON)
.build();

given().spec(spec).when().get("/users").then().statusCode(200);
```

Q. How do you test an API that requires query parameters and path parameters?

Use `.queryParams()` (or `.queryParams()` for multiple) for URL query strings, and `.pathParam()` for placeholders embedded in the endpoint template.

```
given()
.pathParam("id", 5)
.queryParam("expand", "profile")
.when()
.get("/users/{id}")
.then()
.statusCode(200);
```

Q. How would you set and validate custom request/response headers?

Add headers via `.header()/.headers()` on the request side, and validate response headers using `.header(name, matcher)` in the `then()` block — useful for verifying things like caching directives, content-type correctness, or rate-limit headers.

```
given().when().get("/users").then()
.header("Content-Type", containsString("application/json"))
.header("X-RateLimit-Remaining", notNullValue());
```

Q. How do you manage different base URLs/environments (dev, staging, prod) in a Rest Assured framework?

Externalize the base URI into a config/properties file (or environment variables) read at runtime, and select the value based on a passed system property or CI pipeline variable, so the same test suite can run against any environment without code changes.

```
RestAssured.baseURI = System.getProperty("env.baseUrl", "https://staging.api.example.com");
```

Section C | Validation Strategies & Response Handling

Q. How do you validate a JSON response against a predefined schema in Rest Assured?

Use JSON Schema Validation (via the rest-assured json-schema-validator module) to assert the response structure matches a schema file, catching structural regressions (missing fields, wrong types) even when individual value assertions would still pass.

```
given().when().get("/users/1").then()  
.body(matchesJsonSchemaInClasspath("user-schema.json"));
```

Q. How would you validate response time / basic performance expectations within a functional API test?

Rest Assured allows asserting the response time directly, useful as a lightweight guardrail in functional suites — though it should not replace a dedicated performance testing tool for real load analysis.

```
given().when().get("/users").then()  
.time(lessThan(2000L), TimeUnit.MILLISECONDS);
```

Q. How do you handle and validate XML responses with Rest Assured?

Rest Assured supports XML the same way as JSON via `.body()` assertions using XPath-like expressions instead of `JsonPath`, since it auto-detects the content type and applies the appropriate path engine.

```
given().when().get("/legacy/users").then()  
.body("users.user[0].name", equalTo("John Doe"));
```

Q. How would you test negative/error scenarios, such as invalid input or unauthorized access?

Deliberately send malformed payloads, missing required fields, invalid IDs, or omit auth headers, and assert that the API responds with the correct status code (400, 401, 404, etc.) and a meaningful error message — negative testing is often weighted heavily in API test interviews since it reflects real defect-finding ability, not just happy-path coverage.

Q. How do you deserialize a JSON response directly into a Java POJO for assertions?

Call `response.as(SomeClass.class)` (Rest Assured uses Jackson/Gson under the hood) to map the response body into a Java object, then use standard Java assertions against its fields — useful when a response is complex and you want type-safe access rather than repeated `JsonPath` string expressions.

```
User user = given().when().get("/users/1").then()  
.extract().as(User.class);  
assertEquals("John Doe", user.getName());
```

Section D | Framework Design, BDD & CI Integration

Q. How would you structure a Rest Assured framework for maintainability across many endpoints?

Group requests by resource/service into dedicated 'API client' classes (e.g., UserApiClient, OrderApiClient) that wrap Rest Assured calls behind readable methods, keep request specifications and base config centralized, and separate test data/POJOs from test logic — mirroring the separation-of-concerns benefit that Page Object Model gives UI tests.

Q. How do you combine Rest Assured with Cucumber for BDD-style API testing?

Write Gherkin feature files describing API behavior in business language, and implement the step definitions using Rest Assured calls underneath, so stakeholders can read the scenarios while the automation still benefits from Rest Assured's fluent HTTP assertions.

```
Given("a user requests the profile for id {int}", (Integer id) -> {
    response = given().pathParam("id", id).when().get("/users/{id}");
});

Then("the response status should be {int}", (Integer status) -> {
    response.then().statusCode(status);
});
```

Q. How would you run data-driven API tests against many input combinations?

Use TestNG's `@DataProvider` (or JUnit 5's `@ParameterizedTest`) to feed multiple request payloads/expected outcomes into the same test method, or externalize the data into a CSV/JSON file read at runtime, so adding a new test case doesn't require new test code.

Q. How do you integrate Rest Assured API tests into a CI/CD pipeline?

Package tests with Maven/Gradle, trigger via the pipeline tool on PR/merge events, run in headless (no browser needed for pure API tests, so these are typically fast), and publish TestNG/JUnit XML results plus any extracted logs/HTML reports as build artifacts — API suites are often run earlier in the pipeline than UI suites since they're faster and catch backend regressions sooner.

Q. How would you test API idempotency or verify that a DELETE followed by a repeat DELETE behaves correctly?

Call the endpoint once and assert the expected success status/response, then call the same DELETE again and assert the correct subsequent behavior (commonly 404 Not Found, since the resource no longer exists) — this kind of state-transition testing is a common practical interview scenario for API QA roles.

Appendix | Quick-Reference Cheat Sheet

Task	Code Snippet
Basic GET	<code>given().when().get("/path").then().statusCode(200);</code>
POST with JSON body	<code>given().contentType(JSON).body(obj).post("/path");</code>
Path parameter	<code>given().pathParam("id", 5).get("/users/{id}");</code>
Query parameter	<code>given().queryParams("limit", 10).get("/users");</code>
JSON path assertion	<code>then().body("data[0].id", equalTo(1));</code>
Schema validation	<code>then().body(matchesJsonSchemaInClasspath("x.json"));</code>
Extract value	<code>response.jsonPath().getString("id");</code>
Deserialize to POJO	<code>response.as(User.class);</code>
Basic auth	<code>given().auth().preemptive().basic(user, pass);</code>
Bearer token	<code>given().header("Authorization", "Bearer " + token);</code>
Response time check	<code>then().time(lessThan(2000L), MILLISECONDS);</code>
Reusable spec	<code>new RequestSpecBuilder().setBaseUri(...).build();</code>

Final Interview Tip: Interviewers frequently ask you to design negative/edge-case tests, not just happy-path calls — practice narrating how you'd test invalid input, missing auth, and boundary values for any endpoint they describe.