API Automation Interview Questions - Java & Frameworks
By Ranjit Appukutti

REST API Fundamentals (5 Questions)

1. What are the key differences between REST and SOAP?

Answer: REST uses JSON/XML over HTTP with standard methods (GET, POST, PUT, DELETE), is stateless, and lightweight. SOAP uses only XML with WSDL contracts, supports WS-Security, and is protocol-independent. REST is easier to implement and more popular for modern APIs.

2. Explain HTTP status codes commonly used in API testing

Answer:

- **2xx Success:** 200 (OK), 201 (Created), 204 (No Content)
- **3xx Redirection:** 301 (Moved Permanently), 302 (Found)
- **4xx Client Error:** 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found)
- **5xx Server Error:** 500 (Internal Server Error), 503 (Service Unavailable)

3. What is idempotency in REST APIs?

Answer: An idempotent operation produces the same result regardless of how many times it's executed. GET, PUT, DELETE are idempotent. POST is not idempotent as multiple requests create multiple resources.

4. Difference between PUT and PATCH?

Answer: PUT replaces the entire resource, requires sending all fields. PATCH partially updates a resource, only modified fields need to be sent.

5. What is API versioning and common strategies?

Answer: API versioning manages changes without breaking existing clients. Strategies include:

- URI versioning: `/api/v1/users`
- Header versioning: `Accept: application/vnd.api.v1+json`
- Query parameter: `/api/users?version=1`

REST Assured Framework (8 Questions)

6. Write a basic GET request using REST Assured

```
  import io.restassured.RestAssured;
  import static io.restassured.RestAssured.*;
  import static org.hamcrest.Matchers.*;

 @Test
 public void testGetUser() {
     given()
         .baseUri("https://api.example.com")
         .header("Content-Type",  "application/json")
     .when()
         .get("/users/1")
     .then()
         .statusCode(200)
         .body("name", equalTo("John"))
         .body("email", notNullValue());
 }
```

7. How to perform POST request with request body?

```
@Test
public void testCreateUser() {
    String requestBody = "{ \"name\": \"Alice\", \"email\": \"alice@test.com\" }";


given()
    .contentType(ContentType.JSON)
    .body(requestBody)
.when()
    .post("/users")
.then()
    .statusCode(201)
    .body("id", notNullValue());



      }
```

8. How to extract response values in REST Assured?

```
@Test
public void testExtractResponse() {
    Response response = given()
        .get("/users/1")
    .then()
        .extract().response();

 String name = response.path("name");
 int userId = response.jsonPath().getInt("id");

  assertEquals("John", name)
```

9. How to handle authentication in REST Assured?

```
// Basic Auth
given()
    .auth().basic("username", "password")
.when()
    .get("/secure/resource");

// Bearer Token
given()
    .auth().oauth2("ACCESS_TOKEN")
.when()
    .get("/protected/resource");


// API Key
given()
    .header("X-API-Key", "your-api-key")
.when()
    .get("/api/data");
```

10. Explain RequestSpecification and ResponseSpecification

```
// Request Specification - Reusable request config
RequestSpecification requestSpec = new RequestSpecBuilder()
    .setBaseUri("https://api.example.com")
    .setContentType(ContentType.JSON)
    .addHeader("Authorization", "Bearer token")
    .build();


// Response Specification - Reusable assertions
ResponseSpecification responseSpec = new ResponseSpecBuilder()
    .expectStatusCode(200)
    .expectContentType(ContentType.JSON)
    .build();


  // Usage
  given()
      .spec(requestSpec)
  .when()
      .get("/users")
  .then()
      .spec(responseSpec);
```

11. How to validate JSON schema in REST Assured?

```
@Test
public void testJsonSchema() {
    given()
        .get("/users/1")
    .then()
        .assertThat()
        .body(matchesJsonSchemaInClasspath("user-schema.json"));
}
```

12. How to handle query parameters and path parameters?

```
// Query Parameters
given()
    .queryParam("page", 1)
    .queryParam("limit", 10)
.when()
    .get("/users");


  // Path Parameters
  given()
      .pathParam("id", 123)
  .when()
      .get("/users/{id}");



 // Multiple params
 given()
    .pathParam("userId", 1)
    .pathParam("postId", 5)
 .when()
    .get("/users/{userId}/posts/{postId}");
```

13. How to log request and response details?

```java
given()
    .log().all()  // Logs everything
    .log().headers()  // Only headers
    .log().body()  // Only body
.when()
    .get("/users")
.then()
    .log().ifError()  // Log only if error
    .log().status();  // Log status code
```

Java API Testing Concepts (7 Questions)

14. Write a POJO class for serialization/deserialization

```java
public class User {
    private int id;
    private String name;
    private String email;

    // Constructors
    public User() {}

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}

// Usage
User user = new User("John", "john@test.com");
given()
    .contentType(ContentType.JSON)
    .body(user)
.when()
```

```
        .post("/users");

    // Deserialization
    User responseUser = given()
        .get("/users/1")
        .as(User.class);
```

15. How to use Jackson/Gson for JSON parsing?

```
// Jackson
import com.fasterxml.jackson.databind.ObjectMapper;

ObjectMapper mapper = new ObjectMapper();
String json = mapper.writeValueAsString(user);  // Serialize
User user = mapper.readValue(jsonString, User.class);  // Deserialize

// Gson
import com.google.gson.Gson;


Gson gson = new Gson();
String json = gson.toJson(user);
User user = gson.fromJson(jsonString, User.class);
```

16. Implement a BaseTest class for API tests

```java
public class BaseTest {
    protected static RequestSpecification requestSpec;

    @BeforeClass
    public static void setup() {
        RestAssured.baseURI = ConfigReader.getProperty("base.uri");
        RestAssured.basePath = "/api/v1";

        requestSpec = new RequestSpecBuilder()
            .setContentType(ContentType.JSON)
            .addHeader("Authorization", "Bearer " + getToken())
            .setRelaxedHTTPSValidation()
            .build();
    }

    @BeforeMethod
    public void beforeMethod() {
        RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();
    }

    private static String getToken() {
        // Token generation logic
        return "sample_token";
    }
}
```

17. How to implement data-driven testing for APIs?

```
@DataProvider(name = "userData")
public Object[][] getUserData() {
    return new Object[][] {
        {"user1", "user1@test.com"},
        {"user2", "user2@test.com"},
        {"user3", "user3@test.com"}
    };
}



@Test(dataProvider = "userData")
public void testCreateMultipleUsers(String name, String email) {
    User user = new User(name, email);




given()
    .body(user)
.when()
    .post("/users")
.then()
    .statusCode(201)
    .body("name", equalTo(name));
```

18. Explain TestNG groups for API test organization

```
@Test(groups = {"smoke"})
public void testGetAllUsers() {
   // Smoke test
}

@Test(groups = {"regression", "user"})
public void testCreateUser() {
   // Regression test
}

@Test(groups = {"regression", "user"}, dependsOnMethods = {"testCreateUser"})
public void testUpdateUser() {
   // Dependent test
}

// In testng.xml
<groups>
   <run>
      <include name="smoke"/>
   </run>
</groups>
```

19. How to handle file uploads in API testing?

```
  @Test
  public void testFileUpload() {
     File file = new File("src/test/resources/test-file.pdf");
  given()
     .multiPart("file", file, "application/pdf")
     .multiPart("description", "Test document")
  .when()
     .post("/upload")
  .then()
     .statusCode(200)
     .body("fileName",  equalTo("test-file.pdf"));
```

20. Implement retry logic for flaky API tests

```
  public class RetryAnalyzer implements IRetryAnalyzer {
      private int retryCount = 0;
      private static final int MAX_RETRY = 3;


     @Override
     public boolean retry(ITestResult result) {
         if (retryCount < MAX_RETRY) {
             retryCount++;
             return true;
         }
         return false;
     } }

// Usage
@Test(retryAnalyzer = RetryAnalyzer.class)
public void testFlakySAPI() {
    // Test code
}
```

Framework Design & Best Practices (8 Questions)

21. Design a config.properties management class

```
 public class ConfigReader {
     private static Properties properties;

    static {
        try {
            FileInputStream fis = new FileInputStream("src/test/resources/config.properties");
            properties = new Properties();
            properties.load(fis);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static String getProperty(String key) {
        return properties.getProperty(key);
    }
```

22. Implement singleton pattern for API client?

```
public class APIClient {
    private static APIClient instance;
    private RequestSpecification requestSpec;

    private APIClient() {
        requestSpec = new RequestSpecBuilder()
            .setBaseUri(ConfigReader.getProperty("base.uri"))
            .setContentType(ContentType.JSON)
            .build();
    }

    public static APIClient getInstance() {
        if (instance == null) {
            synchronized (APIClient.class) {
                if (instance == null) {
                    instance = new APIClient();
                }
            }
        }
        return instance;
    }

    public RequestSpecification getRequestSpec() {
        return requestSpec;
    }
}
```

23. Create a response validator utility

```
public class ResponseValidator {

public static void validateStatusCode(Response response, int expectedCode) {
    assertEquals(response.getStatusCode(), expectedCode,
        "Status code mismatch");
}

public static void validateResponseTime(Response response, long maxTime) {
    assertTrue(response.getTime() < maxTime,
        "Response time exceeded: " + response.getTime());
}

public static void validateField(Response response, String path, Object expectedValue) {
    Object actualValue = response.path(path);
    assertEquals(actualValue, expectedValue,
        "Field validation failed for: " + path);
}
```

```
public static void validateSchema(Response response, String schemaPath) {
    response.then().assertThat()
        .body(matchesJsonSchemaInClasspath(schemaPath));
}
```

24. Design endpoint management using Enums

```
public enum APIEndpoints {
    GET_USERS("/users"),
    GET_USER_BY_ID("/users/{id}"),
    CREATE_USER("/users"),
    UPDATE_USER("/users/{id}"),
    DELETE_USER("/users/{id}"),
    GET_POSTS("/posts");

    private String endpoint;

     APIEndpoints(String endpoint) {
         this.endpoint = endpoint;
     }


    public String getEndpoint() {
        return endpoint;
    } }

// Usage
given()
    .pathParam("id", 1)
.when()
```

25. Implement extent reports for API tests

```
public class ExtentManager {
    private static ExtentReports extent;
    private static ExtentTest test;

    public static void initReports() {
        ExtentSparkReporter spark = new ExtentSparkReporter("reports/api-test-report.html");
        extent = new ExtentReports();
        extent.attachReporter(spark);
    }

    public static void createTest(String testName) {
        test = extent.createTest(testName);
    }

    public static void logPass(String message) {
        test.log(Status.PASS, message);
    }

    public static void logFail(String message) {
        test.log(Status.FAIL, message);
    }

    public static void flushReports() {
        extent.flush();
    }
}
```

26. Create a reusable API helper class

```
public class APIHelper {

    public static Response get(String endpoint) {
        return given()
            .spec(APIClient.getInstance().getRequestSpec())
        .when()
            .get(endpoint)
        .then()
            .extract().response();
    }
```

```
        public static Response post(String endpoint, Object body) {
            return given()
                .spec(APIClient.getInstance().getRequestSpec())
                .body(body)
            .when()
                .post(endpoint)
            .then()
                .extract().response();
        }

        public static Response put(String endpoint, Object body, String pathParam, Object value) {
            return given()
                .spec(APIClient.getInstance().getRequestSpec())
                .body(body)
                .pathParam(pathParam, value)
            .when()
                .put(endpoint)
            .then()
                .extract().response();
        }

        public static Response delete(String endpoint, String pathParam, Object value) {
            return given()
                .spec(APIClient.getInstance().getRequestSpec())
                .pathParam(pathParam, value)
            .when()
                .delete(endpoint)
            .then()
                .extract().response();
        }
    }
```

27. Implement environment-based test execution

```
    public class EnvironmentManager {
        private static String environment;

        static {
            environment = System.getProperty("env", "qa");
            loadEnvironmentConfig();
        }

        private static void loadEnvironmentConfig() {
            String configFile = "config-" + environment + ".properties";
            // Load specific environment properties
            switch(environment.toLowerCase()) {
```

```
            case "dev":
                RestAssured.baseURI = "https://dev-api.example.com";
                break;
            case "qa":
                RestAssured.baseURI = "https://qa-api.example.com";
                break;
            case "prod":
                RestAssured.baseURI = "https://api.example.com";
                break;
        }
    }

    public static String getEnvironment() {
        return environment;
    }
}

// Run with: mvn test -Denv=qa
```

28. Design page object model for API endpoints

```
public class UserAPI {
    private static final String BASE_PATH = "/users";

    public Response getAllUsers() {
        return APIHelper.get(BASE_PATH);
    }

    public Response getUserById(int userId) {
        return given()
            .pathParam("id", userId)
        .when()
            .get(BASE_PATH + "/{id}")
        .then()
            .extract().response();
    }

    public Response createUser(User user) {
        return APIHelper.post(BASE_PATH, user);
    }

    public Response updateUser(int userId, User user) {
        return APIHelper.put(BASE_PATH + "/{id}", user, "id", userId);
    }

    public Response deleteUser(int userId) {
        return APIHelper.delete(BASE_PATH + "/{id}", "id", userId);
    }
}

// Test class
public class UserAPITest extends BaseTest {
    UserAPI userAPI = new UserAPI();

    @Test
    public void testGetAllUsers() {
        Response response = userAPI.getAllUsers();
        assertEquals(response.getStatusCode(), 200);
    }
}
```

Advanced Scenarios (7 Questions)

29. How to test pagination in APIs?

```
@Test
public void testPagination() {
    int totalPages = given()
        .queryParam("page", 1)
        .queryParam("limit", 10)
    .when()
        .get("/users")
    .then()
        .statusCode(200)
        .extract().path("totalPages");

    for (int page = 1; page <= totalPages; page++) {
        Response response = given()
            .queryParam("page", page)
            .queryParam("limit", 10)
        .when()
            .get("/users")
        .then()
            .statusCode(200)
            .body("data", hasSize(lessThanOrEqualTo(10)))
            .extract().response();

        List<Integer> userIds = response.jsonPath().getList("data.id");
        assertTrue(userIds.size() > 0);
    }
}
```

30. Implement chaining of API requests

```
@Test
public void testAPIChaining() {
    // Step 1: Create user
    User newUser = new User("Chain User", "chain@test.com");
    int userId = given()
        .body(newUser)
    .when()
        .post("/users")
    .then()
        .statusCode(201)
        .extract().path("id");

    // Step 2: Get created user
```

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

        // Step 3: Update user
        newUser.setName("Updated Chain User");
        given()
            .pathParam("id", userId)
            .body(newUser)
        .when()
            .put("/users/{id}")
        .then()
            .statusCode(200);

        // Step 4: Delete user
        given()
            .pathParam("id", userId)
        .when()
            .delete("/users/{id}")
        .then()
            .statusCode(204);
    }
```

31. How to test rate limiting?

```
@Test
public void testRateLimit() {
    int requestCount = 0;
    int maxRequests = 100;
    int rateLimitStatus = 0;



for (int i = 0; i < maxRequests + 10; i++) {
    Response response = given()
        .get("/users");


     requestCount++;


 if (response.getStatusCode() == 429) {
     rateLimitStatus = 429;
     System.out.println("Rate limit hit after " + requestCount + " requests");
```

```
        }
        }

assertEquals(rateLimitStatus, 429, "Rate limit should be enforced");

        }
```

32. Test API with mock server (WireMock)

```
import com.github.tomakehurst.wiremock.WireMockServer;
import static com.github.tomakehurst.wiremock.client.WireMock.*;

public class MockAPITest {
    private WireMockServer wireMockServer;

    @BeforeMethod
    public void setup() {
        wireMockServer = new WireMockServer(8080);
        wireMockServer.start();
        configureFor("localhost", 8080);

        // Mock response
        stubFor(get(urlEqualTo("/users/1"))
            .willReturn(aResponse()
                .withStatus(200)
                .withHeader("Content-Type",  "application/json")
                .withBody("{\"id\":1,\"name\":\"Mock User\"}")));
    }

    @Test
    public void testMockAPI() {
        given()
            .baseUri("http://localhost:8080")
        .when()
            .get("/users/1")
        .then()
            .statusCode(200)
            .body("name", equalTo("Mock User"));
    }

    @AfterMethod
```

```
        public void tearDown() {
            wireMockServer.stop();
        }
    }
```

33. Handle dynamic JSON responses with JsonPath

```
@Test
public void testDynamicJsonParsing() {
    Response response = given()
        .get("/users");

// Extract all user names
List<String> names = response.jsonPath().getList("data.name");

// Extract users with specific condition
List<Map<String, ?>> activeUsers = response.jsonPath()
    .getList("data.findAll { it.status == 'active' }");

// Extract nested values
String firstUserEmail = response.jsonPath()
    .getString("data[0].contact.email");

// Sum of values
int totalAge = response.jsonPath()
    .getInt("data.age.sum()");

assertFalse(names.isEmpty());
assertTrue(activeUsers.size() > 0);

    }
```

34. Implement parallel test execution

```
// testng.xml
<suite name="API Test Suite" parallel="methods" thread-count="5">
    <test name="API Tests">
        <classes>
            <class name="com.tests.UserAPITest"/>
            <class name="com.tests.PostAPITest"/>
        </classes>
    </test>
</suite>

// Thread-safe test implementation
public class ParallelAPITest extends BaseTest {
    private ThreadLocal<User> userData = new ThreadLocal<>();

    @BeforeMethod
    public void setupTestData() {
        userData.set(new User("User" + Thread.currentThread().getId(),
                            "user" + System.currentTimeMillis() + "@test.com"));
    }

    @Test
    public void testCreateUser() {
        given()
            .body(userData.get())
        .when()
            .post("/users")
        .then()
            .statusCode(201);
    }

    @AfterMethod
    public void cleanup() {
        userData.remove();
    }
}
```

35. Create end-to-end API test scenario

```
public class E2EUserJourneyTest extends BaseTest {
    private int userId;
    private int postId;

    @Test(priority = 1)
    public void step1_RegisterUser() {
        User user = new User("E2E User", "e2e@test.com");

        userId = given()
            .body(user)
        .when()
            .post("/auth/register")
        .then()
            .statusCode(201)
            .body("email", equalTo("e2e@test.com"))
            .extract().path("id");

        assertNotNull(userId);
    }

    @Test(priority = 2, dependsOnMethods = "step1_RegisterUser")
    public void step2_LoginUser() {
        String token = given()
            .body(Map.of("email", "e2e@test.com", "password", "password123"))
        .when()
```

```
            .post("/auth/login")
        .then()
            .statusCode(200)
            .extract().path("token");

        // Store token for subsequent requests
        requestSpec.header("Authorization", "Bearer " + token);
    }

    @Test(priority = 3, dependsOnMethods = "step2_LoginUser")
    public void step3_CreatePost() {
        Map<String, Object> post = Map.of(
            "userId", userId,
            "title", "My First Post",
            "body", "This is test content"
        );

        postId = given()
            .spec(requestSpec)
            .body(post)
        .when()
            .post("/posts")
        .then()
            .statusCode(201)
            .extract().path("id");
    }

    @Test(priority = 4, dependsOnMethods = "step3_CreatePost")
    public void step4_GetUserPosts() {
        given()
            .spec(requestSpec)
            .pathParam("userId", userId)
        .when()
            .get("/users/{userId}/posts")
        .then()
            .statusCode(200)
            .body("size()", greaterThan(0))
            .body("find { it.id == " + postId + " }.title", equalTo("My First Post"));
    }

    @Test(priority = 5, dependsOnMethods = "step4_GetUserPosts")
    public void step5_Cleanup() {
        // Delete post
        given()
            .spec(requestSpec)
            .pathParam("id", postId)
        .when()
```

```
            .delete("/posts/{id}")
        .then()
            .statusCode(204);

        // Delete user
        given()
            .spec(requestSpec)
            .pathParam("id", userId)
        .when()
            .delete("/users/{id}")
        .then()
            .statusCode(204);
    }
}
```

Summary

These 35 questions cover:

- REST API fundamentals
- REST Assured framework essentials
- Java concepts for API testing
- Framework design patterns
- Advanced testing scenarios
- Real-world implementation examples

Each answer includes practical code samples that can be directly used in interviews or projects.