API Testing

General API Testing Concepts

1. **What is API testing, and why is it important?**
    - API testing involves testing the application programming interfaces (APIs) directly to ensure they meet functionality, reliability, performance, and security expectations.
    - Importance: Ensures seamless communication between systems, detects issues early, and improves overall application quality.
2. **What are the key differences between API testing and UI testing?**
    - **API Testing**: Focuses on verifying the business logic and data exchange through APIs without a user interface.
    - **UI Testing**: Tests the graphical interface and user interaction, ensuring proper display and functionality.
3. **What is the difference between SOAP and REST APIs?**
    - **SOAP**: Protocol-based, uses XML, and supports stateful operations.
    - **REST**: Architectural style, lightweight, commonly uses JSON, and is stateless.
4. **What are HTTP methods, and where are they used in API testing?**
    - **GET**: Retrieve data.
    - **POST**: Create new data.
    - **PUT**: Update existing data.
    - **DELETE**: Remove data.
    - **PATCH**: Partially update data.
5. **Main components of an HTTP request:**
    - **URL**: Specifies the endpoint.
    - **Headers**: Contains metadata (e.g., authorization, content type).
    - **Body**: Used in methods like POST and PUT.
    - **Method**: HTTP method (e.g., GET, POST).
6. **Key components of an HTTP response:**
    - **Status Code**: Indicates the request's success or failure (e.g., 200 OK, 404 Not Found).
    - **Headers**: Response metadata.
    - **Body**: Contains the response data.
    - **Cookies**: Stores session data.

Understanding Requests and Responses

1. **How do you validate an API response?**
    - Validate:

        - **Status code** matches expected (e.g., 200 for success).
        - **Response time** is within acceptable limits.
        - **Headers** contain required attributes (e.g., content-type).
        - **Body** structure matches the schema and expected data.
2. **Common HTTP status codes:**
    - **200**: Success.
    - **201**: Resource created.
    - **400**: Bad request.
    - **401**: Unauthorized.
    - **403**: Forbidden.
    - **404**: Not found.
    - **500**: Internal server error.
3. **How do you handle authentication and authorization in API testing?**
    - Methods:
        - **Basic Auth**: Username and password.
        - **OAuth**: Token-based authorization.
        - **API Keys**: Key passed in headers or query parameters.
        - **JWT (JSON Web Token)**: Encoded payload with a signature.
4. **Testing API request headers:**
    - Ensure headers like `Authorization`, `Content-Type`, and `Accept` are correctly set.
    - Test invalid, missing, or improperly formatted headers.

Practical API Testing Techniques

1. **Steps involved in API testing:**
    - Analyze requirements.
    - Define test cases.
    - Send requests using tools like Postman.
    - Validate responses.
    - Report issues.
2. **API payload and validation:**
    - Payload: Data sent in the request body (e.g., JSON, XML).
    - Validate payload format, data types, and constraints.
3. **Difference between manual and automated API testing:**
    - Manual: Performed using tools like Postman.
    - Automated: Scripts written using libraries like Rest Assured, with faster execution and CI integration.
4. **Parameterization in API testing:**
    - Testing with different data sets (e.g., query parameters, request bodies).
5. **Schema validation:**
    - Ensures the API response adheres to a defined JSON or XML schema.

Tools for API Testing

1. **Tools for API testing:**
    - Postman, Rest Assured, SoapUI, JMeter, Swagger, Curl.
2. **Postman usage:**
    - Create and execute requests.
    - Validate responses.
    - Save requests in collections for reuse.
3. **Collections in Postman:**
    - Grouping of requests for organizing and automating tests.
4. **Postman vs. Rest Assured:**
    - **Postman**: GUI-based, easy for manual testing.
    - **Rest Assured**: Code-based, suitable for automation and integration.

Advanced Concepts

1. **API mocking:**
    - Creating a simulated API when the real API is unavailable for testing.
2. **Testing negative scenarios:**
    - Sending invalid inputs, missing headers, unauthorized access.
3. **Rate limiter testing:**
    - Simulate excessive requests to ensure the server enforces limits.
4. **Testing security vulnerabilities:**
    - Input validation (SQL injection, XSS).
    - Verify authentication mechanisms.

Validation Techniques

1. **Validating JSON/XML responses:**
    - Use libraries like `JSONPath` or `XPath`.
2. **Performance validation:**
    - Measure response time using tools like JMeter.
3. **Idempotent API methods:**
    - Multiple identical requests yield the same result (e.g., GET, DELETE).

Scenario-Based Questions

1. **Testing a login API:**
    - Send valid and invalid credentials.
    - Validate tokens or session IDs in the response.
2. **REST API for e-commerce:**
    - Test endpoints like `add-to-cart`, `checkout`.
3. **Incomplete documentation:**
    - Use tools like Swagger for API exploration.
4. **Backward compatibility:**
    - Test existing functionality after API updates.

Best Practices

1. Use detailed logging for API failures.
2. Ensure comprehensive test coverage, including edge cases.
3. Automate repetitive tests and integrate into CI/CD pipelines.
4. Regularly update tests to align with API changes.