**RestAssured**

What is RestAssured ?

REST Assured is a Java-based library that is used for testing RESTful APIs. Its architecture consists of several key components that work together to provide a powerful and flexible testing framework.

1. Request Specification: The Request Specification defines the request that is sent to the API endpoint. It includes information such as the HTTP method, headers, query parameters, and request body.

2. Response Specification: The Response Specification defines the expected response from the API endpoint. It includes information such as the expected HTTP status code, headers, and response body.

3. Filters: Filters are used to modify requests and responses at various points in the testing process. They can be used to add authentication, logging, or other functionality to requests and responses.

4. Matchers: Matchers are used to validate the response received from the API endpoint against the expected response. They can be used to check status codes, headers, and response body content.

5. Validatable Response: The Validatable Response is the result of executing a request against an API endpoint. It is used to perform validation using the Matchers.

6. RestAssured class: The RestAssured class is the entry point to the REST Assured library. It provides static methods that allow users to configure the Request and Response Specifications, add Filters, and perform HTTP requests.

7. Test Frameworks: REST Assured can be integrated with popular Java-based test frameworks such as JUnit, TestNG, and Cucumber. This allows users to easily create and manage test suites for their APIs.

Overall, the REST Assured architecture is designed to provide a flexible and extensible testing framework that can be easily integrated with existing Java-based development and testing tools.

﻿What is folder structure in Test Automation Framework?

The folder structure in a test automation framework depends on the specific framework being used and the requirements of the project. However, here is a general example of a folder structure that is commonly used in test automation frameworks:

1. src/: This folder contains the source code of the automation framework, such as the libraries, utilities, and common functions used by the tests.

2. tests/: This folder contains the actual test cases, organized into subfolders based on their respective features, modules, or components.

3. data/: This folder contains test data in various formats such as JSON, XML, or CSV. It may also contain configuration files for different environments or test suites.

4. reports/: This folder contains the output reports of test runs, including logs, screenshots, and metrics.

5. lib/: This folder contains third-party libraries or dependencies required by the framework or tests.

6. config/: This folder contains configuration files for the automation

framework, such as settings for the test environment, test data, or test execution parameters.

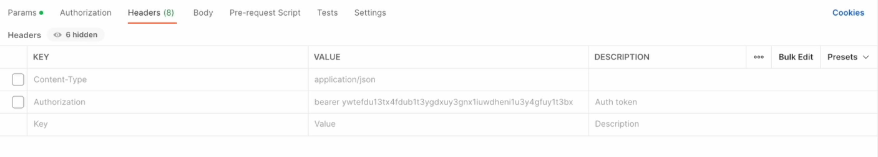
7. resources/: This folder contains any additional resources required by the tests, such as images, videos, or audio files.

8. utils/: This folder contains utility scripts and functions that may be used by the tests, such as date/time functions or file manipulation utilities.

9. drivers/: This folder contains the driver files for different browsers or platforms that are required by the automation framework.

The folder structure may vary depending on the needs of the project and the testing framework used. However, it is essential to have a well-organized and consistent folder structure to make it easier to manage and maintain the test automation framework.

Given() – whatever in Authorization, params, headers and Body goes into this section..



When() –



Then()



Interview Questions:

**1) Explain the difference between 'given()', 'when()', and 'then()' in REST Assured.**

given(): Sets up preconditions (e.g., base URI, headers, parameters).

when(): Specifies the action (HTTP method and endpoint).

then(): Defines expected outcomes (e.g., status code, response body).

**2) What is an API ?**

-> An API, or application programming interface, is a set of rules or protocols that enables software applications to communicate with each other to exchange data, features and functionality.

An API takes the requests from the client and fetch the response from the server.

API testing involves testing these set of rules to ensure that they perform as expected.

**3) what is API Testing ?**

API testing is a process used to validate the performance, security, and usability of APIs.

It aids in finding bugs, problems, and security weaknesses and makes sure the API generates the desired results.

**4) why do we need API Testing ?**

API Testing is crucial for ensuring the quality and reliability of software applications.

API testing is used to validate that the application's functional requirements have been met.

API testing is used to test the ***integration*** between different components of an application, such as between the front-end and back-end.

API testing can also be used to verify the *security* of an application.

***Performance*** Testing: This can help to identify bottlenecks, optimize resource utilization, and ensure that the application meets its performance requirements.

***Documentation***: API testing helps to validate the accuracy of the documentation for the application's API.

**5) Types of Protocols - HTTP and HTTPS ?**

HTTP and HTTPS are communication protocols used to transfer data over the internet.

-> ***HTTP (HyperText Transfer Protocol)*** is the standard protocol for transmitting data over the web. It is used to request and deliver web pages and other content over the internet.

HTTP operates on the client-server model, where a user's browser acts as the client and the website's server acts as the server.

***HTTPS (HyperText Transfer Protocol Secure)*** is a variant of HTTP that is encrypted to provide secure communication over the internet.

In HTTPS, the data sent between the browser and the server is encrypted using SSL (Secure Sockets Layer) or TLS (Transport Layer Security) protocols, which makes it more difficult for unauthorized parties to intercept and read the data.

**6) What is a Monolithic Architecture?**

A monolithic architecture is a traditional approach to designing software where an entire application is built as a single unit. In this architecture, all the different components of the application, such as the user interface, business logic, and data access layer, are tightly integrated and deployed together.

This means that any changes or updates to the application require modifying and redeploying the entire monolith.

Advantages of having a Monolithic Architecture:

1) Simplicity (all the code for your application is in one place)

2) Development Speed

3) Deployment

4) Debugging is easy

Disadvantages:

1) complexity: As a monolithic application grows, it becomes more complex and harder to manage.

2) scalability

3) Technology stack(all parts of the application share the same technology stack)

**7) What is a Microservice Architecture ?**

In a microservices architecture, an application is built as a collection of small, independent services, each representing a specific business capability.

These services are loosely coupled and communicate with each other over a network, often using protocols like HTTP or messaging queues.

Advantages:

1) Scalability: scaled independently based on demand.

2) Flexibility: Microservices enable teams to use different technologies for different services.

3) Resilience: a failure in one service does not necessarily impact the entire application.

Disadvantages:

1) Complexity: Managing a large number of microservices can be complex.

2) Testing: It requires a comprehensive testing strategy that covers integration testing between services, as well as unit testing within each service.

**8) Importance of API Testing in Microservices architecture ?**

API testing is crucial in microservice architecture because APIs serve as the communication layer between the different microservices.

In a microservice architecture, each microservice is responsible for a specific functionality and communicates with other microservices through APIs.

- Integration Testing

- Performance Testing

- Security Testing

**9) what is a Web Service ?**

Web services are a way for different computer systems to communicate and exchange information over the internet.

Webservices require a network to interact and network communication is generally achieved using Simple Object Access Protocol (SOAP). SOAP uses XML format for encoding data. The XML format is used for storing and transferring information using HTTP protocol.

**10) what is an API ?**

An application programming interface (API) is a set of rules that define how applications can communicate with each other. There are different types of API’s like GraphQL, REST, WebSocket APIs, RPC APIs, etc. All APIs have their own set of protocols and definitions. So each API type operates differently.

**11) Difference between Web Service and an API ?**

| **Web Services** | **Web API** |
| --- | --- |
| Web services are a type of API, which must be accessed through a network connection. | APIs are application interfaces, implying that one application can communicate with another application in a standardized manner. |
| Web service is used for REST, SOAP and XML-RPC for communication. | API is used for any style of communication. |
| All Web services are APIs. | APIs are not web services. |
| It provides supports only for the HTTP protocol. | It provides support for the HTTP and HTTPS protocol: URL Request/Response Headers, and so on. |
| Web service supports only XML. | API supports XML and JSON. |

**12) Difference between SOAP and REST ?**

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 2) | SOAP stands for **Simple Object Access Protocol**. | REST stands for **REpresentational State Transfer**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 5) | **JAX-WS** is the java API for SOAP web services. | **JAX-RS** is the java API for RESTful web services. |
| 6) | SOAP **defines standards**to be strictly followed. | REST does not define too much standards like SOAP. |
| 7) | SOAP **requires more bandwidth** and resource than REST. | REST **requires less bandwidth** and resource than SOAP. |
| 8) | SOAP **defines its own security**. | RESTful web services **inherits security measures** from the underlying transport. |
| 9) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |
| 10) | SOAP is **less preferred** than REST. | REST **more preferred** than SOAP. |

**13) REST ?**

REST is a set of architectural principles that uses the HTTP protocol to exchange data in a lightweight and flexible way. It uses HTTP methods such as GET, POST, PUT, and DELETE to exchange data, and can return data in various formats, including XML and JSON.

**Stateless interactions:** Each request from a client (like your phone app) to a server (like a web store) functions independently. The server doesn't need to remember anything about past interactions to respond to a new request. It treats each request as new.

13) **Principles of REST:**

REST has six guiding constraints which must be satisfied if an interface needs to be referred to as RESTful. These principles are:

*Client-server:* Clients request resources, and servers process those requests and send back responses.

*Stateless:* Each request from the client to the server must contain all the information needed to understand and process the request. The server should not store anything about the latest HTTP request the client made. It treats each request as new.

***Cacheable:*** Whenever possible, responses from a server can be cacheable by clients. This can significantly improve performance by reducing the number of requests that need to be sent to the server.

*Uniform Interface:* REST APIs rely on a standardized interface for communication. This interface uses HTTP methods (GET, POST, PUT, DELETE) and URIs (Uniform Resource Identifiers) to interact with resources.

*Layered System:* The architecture can be composed of multiple layers, each one with specific functionality and responsibilities. A layered system architecture allows for intermediaries like caches, load balancers, and security gateways to be inserted between clients and servers.

*Code on Demand (optional):* Servers can provide executable code or scripts for clients to execute in their context.

**14) SOAP ?**

-> SOAP is a protocol for exchanging structured information in web services using XML.

-> SOAP is a messaging protocol specifically designed for exchanging information between computers on a network.

-> It uses XML to encode its messages and is typically used in enterprise environments.

-> SOAP is designed to work on any operating system or any programming language.

-> SOAP is more secure than REST, as it has built-in security features such as encryption and digital signatures.

**15) Synchronous vs Asynchronous API ?**

***Synchronous***: If an API call is synchronous, it means that code execution will block (or wait) for the API call to return before continuing. This means that until a response is returned by the API, your application will not execute any further.

***Asynchronous***: Asynchronous calls do not block (or wait) for the API call to return from the server. Execution continues on in your program, and when the call returns from the server, a “callback” function is executed.

**16) HTTP Methods:**

**GET** method is used to retrieve data from a server. You can use it to retrieve one or more resources from an API.

**POST** is used to create a new resource. It requires the client to send the request body. POST is NOT idempotent. So if you retry the request N times, you will end up having N resources with N different URIs created on server.

**PUT** is used to update a resource completely. It requires the client to send the complete resource representation. The PUT method is idempotent.

**PATCH** is used to partially update a resource. It requires the client to send only the changes.

**DELETE** is used to delete data from a server. It is used to delete a specific resource identified by a unique identifier, such as a user id.

**HEAD** method is similar to GET method, but it only retrieves the headers of the response message, which includes information about the resource such as the content type, size, and last-modified date. The response body is not returned.

**OPTIONS** method is used to retrieve the HTTP methods that the server supports for a specific URL.

Allow: GET, POST, HEAD, OPTIONS

*Idempotency:*

In the context of REST APIs, GET, PUT, and DELETE methods are typically idempotent. This means if you make the same request multiple times, the state of the server should be the same as if you had made the request only once.

16) **What are the types of Status codes?**  
  
1xx informational response – the request was received, continuing process  
2xx successful – the request was successfully received, understood, and accepted  
3xx redirection – further action needs to be taken in order to complete the request  
4xx client error – the request contains bad syntax or cannot be fulfilled  
5xx server error – the server failed to fulfil an apparently valid request

**17) HTTP Status codes ?**

200:  OK  
201:  Created  
204:  No Content  
400:  Bad Request  
401: Unauthorized  
403: Forbidden  
404: Not Found  
302: Redirect  
405: Method Not Allowed  
422: Unprocessable Entity  
417: Expectation failed  
500: Internal Server Error

200 OK: The request was successful.

201 Created: The request was successful, and a resource was created as a result.

400 Bad Request: The server could not understand the request due to invalid syntax.

401 Unauthorized: The client must authenticate itself to get the requested response.

404 Not Found: The server can not find the requested resource.

**18) Details in API documentation ?**

- API endpoints

- Parameters

- HTTP Methods

- Authentication requirements

- Response format

- Error codes and messages

19)

**Collection**: A collection is a group of related requests that can be organized together in Postman. Collections can be used to organize and run tests for multiple API endpoints.

**Runner**: The Runner is a feature in Postman that allows you to execute a collection of requests in a sequence. It provides an easy way to run tests against multiple endpoints and verify the responses.

**Global variable**: Global variables are values that can be defined once and used across multiple requests in a collection.

**Methods**: Methods refer to the HTTP methods that can be used to interact with an API endpoint. Common methods include GET, POST, PUT, DELETE, and PATCH.

**Headers**: Headers are additional pieces of information that can be included with a request or response. They can be used to provide metadata about the request or response, such as the content type, encoding, authentication credentials, and more.

**Body**: The body of a request is the data that is sent with the request. This can include form data, JSON, or XML.

**Params**: Params refer to the query parameters that can be included with a request. These parameters are typically used to filter data or provide additional information to the API.

**Auth**: Auth refers to the authentication method used to secure the API. This can include basic authentication, OAuth, API keys, or other methods.

**Authentication** and **Authorization**: Authentication is the process of verifying the identity of a user or client. Authorization is the process of granting or denying access to a specific resource or action based on the user's identity and permissions. REST APIs typically use different authentication and authorization mechanisms such as OAuth, API keys, basic authentication, and more.

**Cookies**: Cookies are small pieces of data that are sent from a server and stored on the client-side. They can be used to maintain user sessions, store user preferences, and more.

**Query Parameter**: Query parameters are used to filter or limit the data returned by an API. They are usually included in the URL after a question mark (?) and can be used to provide additional information to the API endpoint.

**Path Parameter**: Path parameters are used to identify a specific resource or endpoint within an API. They are usually included in the URL as part of the endpoint path.

**Form Parameter**: Form parameters are used to submit data to an API as part of a form. They are typically sent in the request body and can include different types of data such as text, numbers, files, and more.

**20) Pagination and Filtering ?**

Pagination: When dealing with large amounts of data, it's often not practical to retrieve all resources at once. Pagination allows you to retrieve data in small chunks, which can significantly improve performance for large datasets

Filtering is another technique to limit the amount of data returned by the API.

**21) Rate Limiting ?**

Rate limiting is a technique for limiting network traffic. It sets a limit on how many requests a client can make to the API in a given amount of time. This is crucial for maintaining the health of your API and protecting it from abuse.

**22) Versioning ?**

As your API evolves, you'll likely need to make changes that aren't compatible with existing clients. Versioning allows you to introduce these changes without breaking existing clients. There are several strategies for versioning your API, such as including the version in the URL (e.g., /v1/users) or using a custom HTTP header.

**23) Security ?**

API security is a critical aspect of API design. This includes techniques like using HTTPS to encrypt the data in transit, using authentication and authorization to control access to the API, and validating and sanitizing input to protect against attacks like SQL injection and cross-site scripting (XSS).

**24) What is REST Assured?**

REST Assured is a Java library that provides a domain-specific language (DSL) for writing powerful, maintainable tests for RESTful APIs. With REST Assured, you can write tests in a more descriptive manner. It simplifies the process of validating and verifying the responses of the API, making it a popular choice for testing REST APIs.

**25) structure of a REST Assured test (Given, When, Then)?**

REST Assured tests follow a Given-When-Then structure, which is a common structure for behavior-driven development (BDD):

*Given*: This is where you prepare the API request. You can specify things like the base URI, path parameters, query parameters, headers, and the body of the request.

*When*: This is where you send the API request. You specify the HTTP method (like GET, POST, PUT, DELETE) and the endpoint.

*Then*: This is where you validate the API response. You can check things like the status code, headers, and the body of the response.

**26) Hamcrest matchers ?**

Hamcrest matchers are a powerful tool for writing assertions in your tests.

-> equalTo("value") is a Hamcrest matcher that checks if the actual value is equal to the expected value.

-> not(equalTo("value")) checks if the value of "key" in the response body is not equal to "value".

-> nullValue() checks if the value of "key" in the response body is null.

-> notNullValue() checks if the value of "key" in the response body is not null.

**->** Checking if a list has a certain size: body("listKey.size()", is(3));

-> hasItem("value") checks if the list in the response body contains specified "value".

body("listKey", hasItem("value"))

-> greaterThan(10) checks if the value of "key" in the response body is greater than specified value.

-> Asserting multiple conditions: You can use allOf() , anyOf() , and noneOf() to assert multiple conditions at once:

body("key", allOf(notNullValue(), instanceOf(String.cla ss), equalTo("value")));

checks if the value of "key" in the response body is not null, is an instance of String, and is equal to "value".

-> hasItems() to check if a list contains certain values:

body("listKey", hasItems("value1", "value2", "value 3"));

-> matchesPattern() to check if a value matches a regular expression:

body("key", matchesPattern("[a-z]+")); checks if the value of "key" in the response body matches the regular expression "[a-z]+", which means one or more lowercase letters.

-> ordered() checks if the list in the response body is sorted in ascending order.

**27) Parameterization in REST Assured (Path Param, Query Param):**

Parameters are a way to pass information to the server to influence the response. REST Assured supports both path parameters and query parameters.

-> Path parameters are part of the URL

-> Query parameters are appended to the URL

**28) Validating Response body**

then()

.statusCode(200)

.body(“key”, equalTo(”value”))

.time(lessThan(2000L))

.header("Content-Type", "application/json")

.cookie("cookieName", "cookieValue")

**29) Serialization and Deserialization ?**

Serialization is the process of converting an object into a format that can be easily stored or transmitted and then reconstructed later. In the context of REST APIs, this typically means converting a Java object into a JSON or XML string.

Deserialization is the reverse process - it involves converting a JSON or XML string back into a Java object.

Serialization and deserialization are handled automatically by REST Assured when you use the body() and as() methods.

**30)**

**Jackson (Fasterxml):** Jackson is a suite of data-processing tools for Java including the flagship streaming JSON parser / generator library, matching data-binding library (POJOs to and from JSON).

Here's how you can use Jackson's *ObjectMapper* class to serialize and deserialize objects in REST Assured:

ObjectMapper mapper = new ObjectMapper();

String json = mapper.writeValueAsString(user); //Serialization

ObjectMapper mapper = new ObjectMapper();

User user = mapper.readValue(response, User.class); //Deserialization

**Gson**: Gson is a Java library that can be used to convert Java Objects into their JSON representation and vice versa.

Gson gson = new Gson();

String json = gson.toJson(user); //Serialization

Gson gson = new Gson();

User user = gson.fromJson(response, User.class); //Deserialization

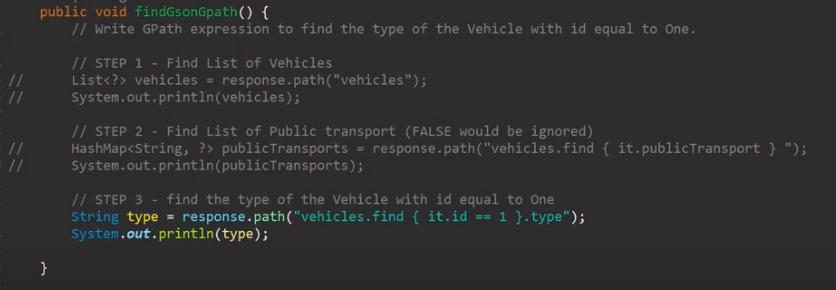
**31) Schema Validation ?**

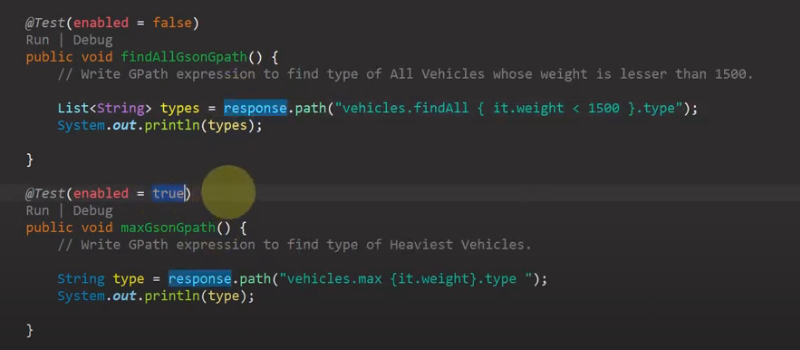
Schema Validation allows us to ensure that the API is returning data in the expected format.

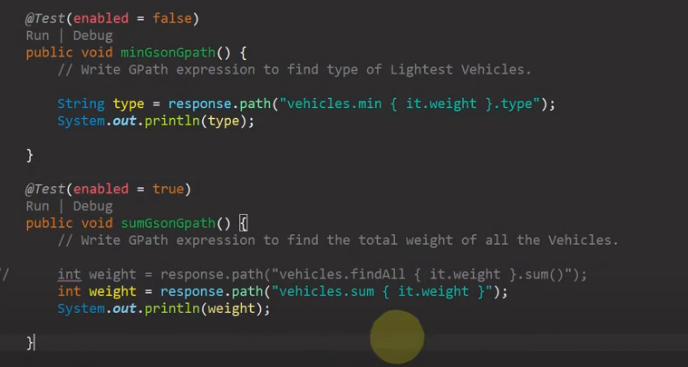
This is typically done by comparing the API response with a predefined schema.

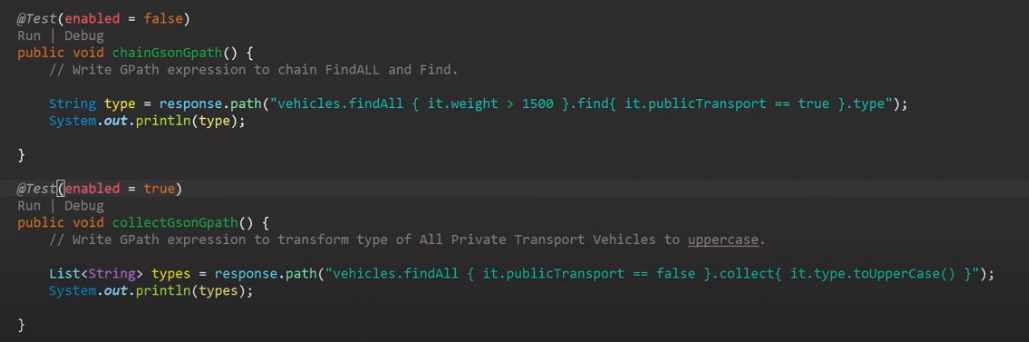
then(). body(*matchesJsonSchemaInClasspath*("user-schema.json")); // checks if the response body matches the "user-schema.json" schema in the classpath.

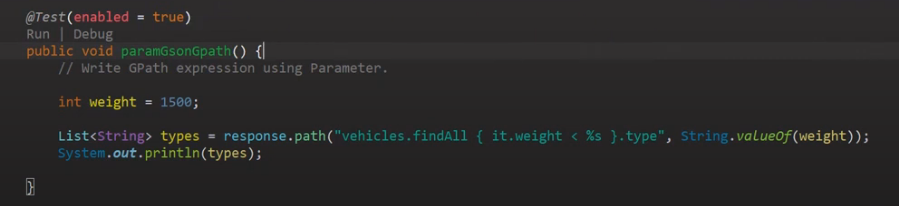
then(). body(matchesXsdInClasspath("user-schema.xsd")); // checks if the response body matches the "user-schema.xsd" schema in the classpath.











Find will return a single value that matches the specified condition

If there is more than one condition satisfied by the close, Find will return the first occurrence only.

use the **find** method with a filter to extract a specific value

path("listKey.find { it == 'value' }"); //finds the first occurrence of "value" in "listKey".

path("employees.find { it.firstName == 'Anna' }.lastName");

We can use findAll to return a new collection with all the results that match the condition.

path("listKey.findAll { it.startsWith('value') }");

path("employees.findAll { it.firstName.startsWith('J') }.firstName");

If you want to do some transformation to each element of a list, and collect the responses in a new list, you can use the collect method:

people.collect { it.toUpperCase() }

min(), max(), sum() methods:

def myList = [2,3,4,4,4,5,5,6]

def sumOfMyListValuesGreaterThanFour = myList.findAll { it > 4 }.sum() // returns 16

def minValueGreaterThanTwo = myList.findAll { it > 2 }.min() // returns 3

def maxValueBetweenThreeAndFive = myList.findAll { it > 3 && it < 5 }.max() // returns 4

**32) Logging:**

given().

log().all().

when().

get("/endpoint").

then().

log().body();

//In this example, log().all() logs all details of the request and log().body() logs the body of the response.

**Continuous Integration:**

Continuous integration (CI) is a practice where developers integrate their changes into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. Tools like Jenkins, Travis CI, or CircleCI can be used to run your REST Assured tests as part of your CI pipeline.