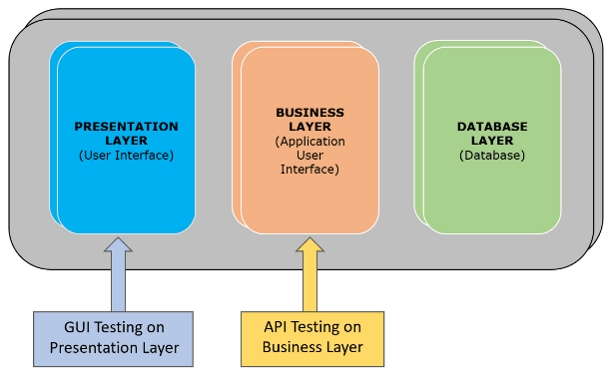
# ­­­API Module

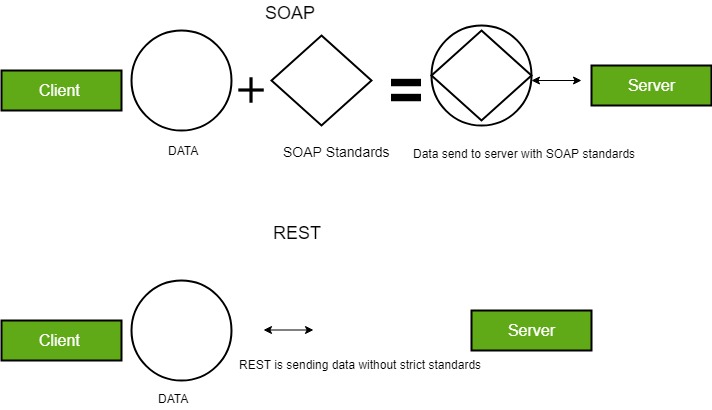
1. What is API:

API (Application Programming Interface) is a computing interface which enables communication and data exchange between two separate software systems.



1. Rest API And SOAP API Differences:

|  |  |  |
| --- | --- | --- |
| S.NO. | SOAP | REST |
| 1 | Simple Object Access Protocol | Representational State Transfer. |
| 2 | Since SOAP is a protocol, it follows a strict standard to allow communication between the client and the server | Is an architectural style that doesn’t follow any strict standard but follows six constraints defined by Roy Fielding in 2000 |
| 3 | It uses only XML for exchanging information in its message format | It’s the choice of implementer which Media-Type to use like XML, JSON, Plain-text. Moreover, REST can use SOAP protocol, but SOAP cannot use REST. |
| 4 | SOAP is difficult to implement and it requires more bandwidth | REST is easy to implement and requires less bandwidth such as smartphones. |
| 5 | SOAP has SSL (Secure Socket Layer) and WS-security | REST has SSL and HTTPS. |
| 6 | The ways APIs exposes the business logics are services interfaces like @WebService. | The ways APIs exposes the business logics are services interfaces like @path("/WeatherService") |



1. Rest Constraints:

* Client – Server Architecture style:

This is the first constraint in the REST architectural style. According to this constraint, an application should be modeled like a Client – Server. To relate to it, application should have the UI separate from the Data. There should be a distinct component handling Front end (UI) and the Back end (Database).

* Stateless:

This is the second constraint in the REST architectural style. According to this constraint a Server should not store context of the Client. Each request sent by the Client should not be treated with any information (context) from previous requests made by the same client.

The importance of this constraints is that because of its Server implementations becomes simple. It now doesn’t need to store references of clients.

* Cache:

According to this constraint Responses from the server should contain relevant information to tell whether the Response can be cached by the client of not. Usually this is done via a Header entry in the Response.

This constraint improves the client efficiency, for cacheable responses Client need not make requests to the server. Client can simply investigate its cache for the Response. This saves network bandwidth and Client processing power.

* Uniform Interface:

Its constraint defines the interface between clients and servers

Roy Fielding (originator of the Rest architectural style) states that the aim of using uniform interfaces is to:

ease and improve global architecture and the visibility of interactions

* Layered System:

According to this constraint the system implementation should be layered. Each layer abstracting out certain functionality of the overall system. A layer should not know about the existence of other layers apart from the layers that it directly interacts with.

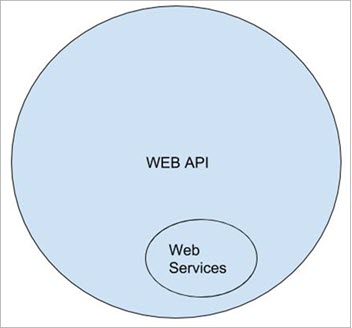
* Code on demand

According to this constraint a client can extend its functionality by downloading code from the Server.

1. API Vs Webservices:

The only difference is that a Web service facilitates interaction between two machines over a network. An API acts as an interface between two different applications so that they can communicate with each other. An API is a method by which the third-party vendors can write programs that interface easily with other programs. A Web service is designed to have an interface that is depicted in a machine-processable format usually specified in Web Service Description Language (WSDL). Typically, “HTTP” is the most used protocol for communication. Web service also uses SOAP, REST, and XML-RPC as a means of communication. API may use any means of communication to initiate interaction between applications. For example, the system calls are invoked using interrupts by the Linux kernel API.

All Web Services are Web APIs, but all Web APIs are not Web Services



Web Service

* Web Services generally use XML (Extensible Markup Language), which means they are more secure.
* Web Services is more secure as both Web Services and APIs provide SSL (Secure Socket Layer) during data transmission, but it also provides WSS (Web Services Security).
* Web Service is a subset of Web API. For Example, Web Services are based only on three styles of use i.e., SOAP, REST and XML-RPC.
* Web Services always need a network to operate.

Web API

* A Web API generally uses JSON (JavaScript Object Notation), which means Web API is faster.
* Web API is faster as JSON is light-weighted, unlike XML.
* Web APIs are the superset of Web Services. For Example, all three styles of Web Services are present in the Web API as well, but apart from that, it uses other styles like JSON – RPC.
* Web API does not necessarily require a network to operate.

1. API Testing:

API is an acronym for Application Programming Interface.

In software application (app) development, API is the middle layer between the presentation (UI) and the database layer. APIs enable communication and data exchange from one software system to another.

API testing is a software testing practice that tests the APIs directly — from their functionality, reliability, performance, to security. Part of integration testing, API testing effectively validates the logic of the build architecture within a short amount of time.

Unit A+B=c > API

1. Types of API Testing:
   * 1. Validation Testing
     2. Functional testing
     3. UI testing
     4. Load Testing
     5. Runtime/Error Detection
     6. Security testing
     7. Penetration Testing
     8. Fuzz testing

Advantages of API Testing:-

* 1. Time Effective:

API Testing doesn't require GUI to be ready and it can be performed way early in the development cycle. The Automated API tests provide much quicker test results and significantly accelerate development workflows; thus, it helps you speed up the feedback loop and catch issues faster.

In addition to that, API tests are significantly less time-consuming when compared to UI Tests. UI Tests spend much time rendering and loading the web pages and interface elements, whereas can execute API tests in seconds. Let's take an example where a user needs to register and login from UI takes at least 3 to 5 minutes, whereas API testing takes less than 30 seconds

* 1. Reduced Cost:

API tests can be executed as early as the business logic is defined and before any GUI testing. So, it will help you to identify the issue at the early stage. Early identification means the less expensive it is to fix it and Reduces the cost of Application changes. API testing enables the QA team to detect and resolve issues before they become a production problem, keeping project costs at bay.

* 1. Language Independent: Select any language for test automation

API tests are Language Independent Since the data is interchanged using JSON or XML and compromised HTTP requests and HTTP responses. So, the QA team is free to choose the language of their choice that supports these technologies ((JavaScript, Java, Ruby, Python, PHP, etc.).

* 1. GUI Independent:

The major core advantage of API testing is that it provides access to applications without users having to interact with a potentially disparate system. This helps the tester to detect and recognize the errors early, instead of them becoming larger issues during GUI testing.

* 1. Improved test coverage:

Unlike unit tests, automated API tests are generally broader in scope. While unit tests are focused on limited functionality of components within a single application, problems often arise at the intersection where the scope of one layer ends and the other begins.

1. Tools used for API Testing:
   1. Postman:
   2. Soap UI
   3. JMeter
   4. Katalon
   5. Tricentis. Etc.
2. Challenges in API Testing:
   1. Test Data Management:

Conventionally UI testing is focused only on the functionality of the overall application. A tester provides input and validates the output against expected outcomes. API testing is like a doorway to data for connecting applications; hence expectations for scenarios/use cases and testing is very high in terms of speed and effectiveness.

* 1. Knowledge of Business Applications logic:

Based on the overall business architecture logic, many business rules are defined on which APIs are developed, integrated, and used. The absence of knowledge and understanding of this business architecture logic and rules among API QA testers lead to uncertainty about the test objectives.

* 1. A sequence of API calls:

In many cases, API calls need to appear in a specific order to work correctly. This creates a sequencing challenge for the testing team.

* 1. Parameter Validation: API Testers also face the problem of validating parameters sent via API requests. Many parameters and options for validation is an extremely difficult task. The tester needs to make sure all parameter data uses the correct string or numerical data type, fits within length restrictions, fits within a designated value range, and passes other validation criteria.
  2. Impact of change: Whenever there is a new version of an API, it will likely cause the entire application to go haywire. As there are multiple dependent components, implementing a change is often highly risky and unpredictable in terms of its effects.

1. Best Practices in API testing:
   1. DRY (Don't Repeat Yourself)

You want to avoid repeating your code, but many tests require you to address the same components or similar actions. In these cases, you can create a common library to wrap your test requests and make the usage shorter and the process simpler.

* 1. Prerequisites and Cleanup
  2. Treat the API you’re Testing Like a Consumer Would
  3. Ensure that all API responses are tracked and saved for posterity.
  4. Make sure to perform “negative” testing.
  5. Remove dependencies when possible

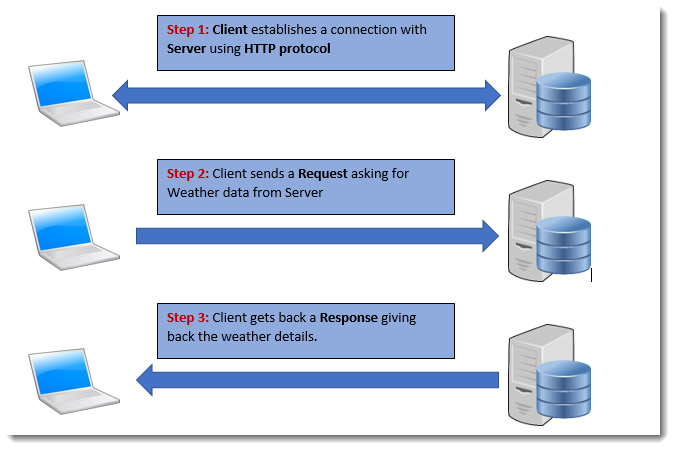
1. API Components:
   1. Server Client Architecture

Ex: You want to know weather of your city.

Graphical user interface, application

Description automatically generated

A Client and a Server establishes a connection using HTTP protocol. Once the connection is established, Client sends across the request to the Server in the form of XML or JSON which both entities (Client and Server) understand. After understanding the request Server responds with appropriate data by sending back a Response.



When a browser requests for a web page, the following set of events takes place:

1. A user enters the URL of the website or file in the browser. The browser then requests the DNS(Domain Name System) server
2. The DNS server looks up the IP address of the web server and once it finds it, sends it back to the browser
3. The browser then sends over an HTTP/HTTPS request to the web server’s IP (provided by the DNS server in the previous step)
4. The server responds with the necessary information or data
5. The browser renders the data and the website is displayed
   1. URLs and URNs

The target of an HTTP request is called a "resource”. it can be a document, a photo, or anything else. Each resource is identified by a Uniform Resource Identifier (URI) used throughout HTTP for identifying resources.

**URLs:** The most common form of URI is the Uniform Resource Locator (URL), which is known as the web address.

Ex: <https://developer.mozilla.org>

<https://developer.mozilla.org/en-US/docs/Learn/>

Any of those URLs can be typed into your browser's address bar to tell it to load the associated page (resource).

**URNs:** A Uniform Resource Name (URN) is a URI that identifies a resource by name in a particular namespace.

Ex: urn:isbn:9780141036144

urn:ietf:rfc:7230

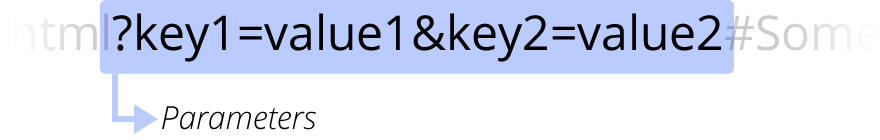
Syntax of URIs:













* 1. Understating of HTTP



Chart, treemap chart

Description automatically generated

* 1. HTTP Caching:

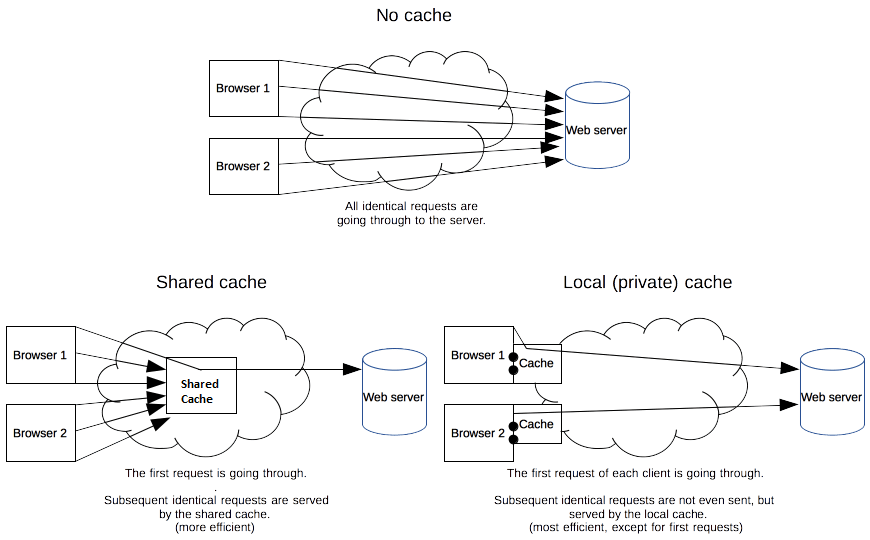
The performance of web sites and applications can be significantly improved by reusing previously fetched resources. Web caches reduce latency and network traffic and thus lessen the time needed to display a representation of a resource. By making use of HTTP caching, Web sites become more responsive.

Advantages of Caching:

* it eases the load of the server that doesn’t need to serve all clients itself
* it improves performance by being closer to the client

Cache types:

* Private Cache
* Shared Cache



**Controlling caching:**

* + 1. Cache-Control: no-store

The cache should not store anything about the client request or server response. A request is sent to the server and a full response is downloaded each and every time

* + 1. Cache-Control: no-cache

A cache will send the request to the origin server for validation before releasing a cached copy.

* + 1. Cache-Control: private
    2. Cache-Control: public
    3. Cache-Control: must-revalidate

When using the "must-revalidate" directive, the cache must verify the status of the stale resources before using it and expired ones should not be used.

* + 1. Cache-Control: max-age=31536000

The most important directive here is max-age=<seconds>, which is the maximum amount of time in which a resource will be considered fresh.

* 1. HTTP Request:

**A correctly composed HTTP request contains the following elements:**

1. **A request line.**
2. **A series of HTTP headers, or header fields.**
3. **A message body, if needed.**
   * 1. HTTP Request URI

Ex: GET /software/htp/cics/index.html HTTP/1.1

* + 1. HTTP Headers

HTTP headers let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon (:), then by its value.

Ex: Accept, Authorization, Content- type

* + 1. HTTP Request Body

The body content of any HTTP message can be referred to as a message body or entity body. Technically, the entity body is the actual content of the message.

* + 1. HTTP Request Methods
       1. Get: This one requests a representation of the specified resource. Requests using GET should only be used to request data (they shouldn't include data).
       2. Post: This method sends data to the server. The type of the body of the request is indicated by the Content-Type header.
       3. Put: This method sends data to the server. The type of the body of the request is indicated by the Content-Type header.

The difference between PUT and POST is that PUT is idempotent: calling it once or several times successively has the same effect (that is no side effect), whereas successive identical POST requests may have additional effects, akin to placing an order several times.

* + - 1. Delete: This request method deletes the specified resource.
      2. Head: This method requests the headers that would be returned if the HEAD request's URL was instead requested with the HTTP GET method.
      3. Options: This method requests permitted communication options for a given URL or server. A client can specify a URL with this method, or an asterisk (\*) to refer to the entire server.
      4. Patch: This request method applies partial modifications to a resource.
    1. HTTP Authentication:

Shape

Description automatically generated with medium confidence

Some common authentication schemes include:

* + - 1. Basic
      2. Bearer
      3. Digest Etc.
  1. HTTP Response:

**An HTTP response contains:**

1. **A status line.**
2. **A series of HTTP headers, or header fields.**
3. **A message body, which is usually needed.**
4. Status Line:

Ex: **HTTP/1.1 200 OK**

* + 1. HTTP Response Status codes
       1. 1XX Codes
       2. 2XX Codes
          1. **200 -OK**

The meaning of a success depends on the HTTP request method:

GET: The resource has been fetched and is transmitted in the message body.

HEAD: The representation headers are included in the response without any message body

POST: The resource describing the result of the action is transmitted in the message body

TRACE: The message body contains the request message as received by the server.

The successful result of a PUT or a DELETE is often not a 200 OK but a 204 No Content (or a 201 Created when the resource is uploaded for the first time).

* + - * 1. **201 Created**

The HTTP 201 Created success status response code indicates that the request has succeeded and has led to the creation of a resource. The new resource is effectively created before this response is sent back and the new resource is returned in the body of the message, its location being either the URL of the request, or the content of the Location header.

* + - * 1. **202 Accepted**

The Hyper Text Transfer Protocol (HTTP) 202 Accepted response status code indicates that the request has been accepted for processing, but the processing has not been completed; in fact, processing may not have started yet.

* + - * 1. **204 No Content**

The HTTP 204 No Content success status response code indicates that a request has succeeded, but that the client doesn't need to navigate away from its current page.

* + - 1. 3XX Codes
         1. **300 Multiple Choices**

The HTTP 300 Multiple Choices redirect status response code indicates that the request has more than one possible responses. The user-agent or the user should choose one of them. As there is no standardized way of choosing one of the responses, this response code is very rarely used.

* + - * 1. **301 Moved Permanently**

The HyperText Transfer Protocol (HTTP) 301 Moved Permanently redirect status response code indicates that the resource requested has been definitively moved to the URL given by the Location headers

Ex:

Client request

GET /index.php HTTP/1.1

Host: [www.example.org](http://www.example.org)

Server response

HTTP/1.1 301 Moved Permanently

Location: [`](http://www.example.org/index.asp)

* + - * 1. **302 Found**

The HyperText Transfer Protocol (HTTP) 302 Found redirect status response code indicates that the resource requested has been temporarily moved to the URL given by the Location header.

* + - 1. 4XX Codes
         1. **400 Bad Request**

This status code indicates that the server cannot or will not process the request due to something that is perceived to be a client error

* + - * 1. **401 Unauthorized**

This client error status response code indicates that the request has not been applied because it lacks valid authentication credentials for the target resource.

* + - * 1. **402 Payment required**
        2. **403 Forbidden**

This client error status response code indicates that the server understood the request but refuses to authorize it.

* + - * 1. **404 Not found**

This client error response code indicates that the server can't find the requested resource. Links that lead to a 404 page are often called broken or dead links.

* + - * 1. **405 Method not allowed**

This response status code indicates that the request method is known by the server but is not supported by the target resource.

* + - 1. 5XX Codes
         1. **500 Internal Server Error**

This server error response code indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

* + - * 1. **503 Unavailable**

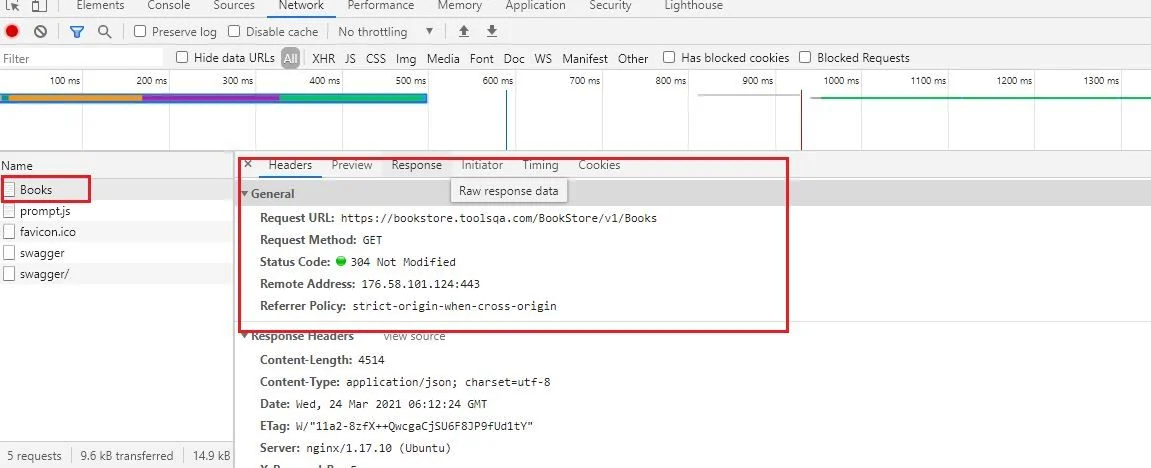
This server error response code indicates that the server is not ready to handle the request.

1. Response Headers
2. Response Body

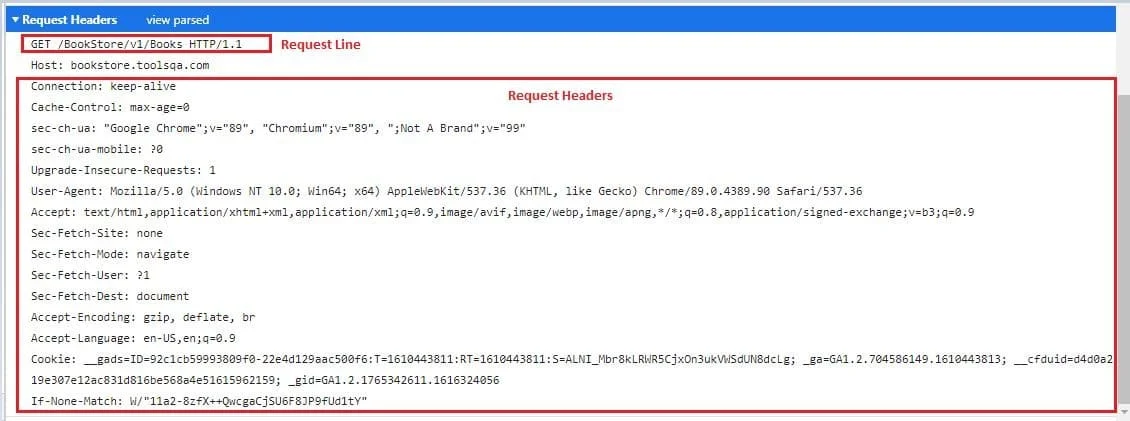
**Rest Assured Module**

* HTTP Request
  1. Request Line
     1. The HTTP method used
     2. The request URI
     3. The HTTP protocol version

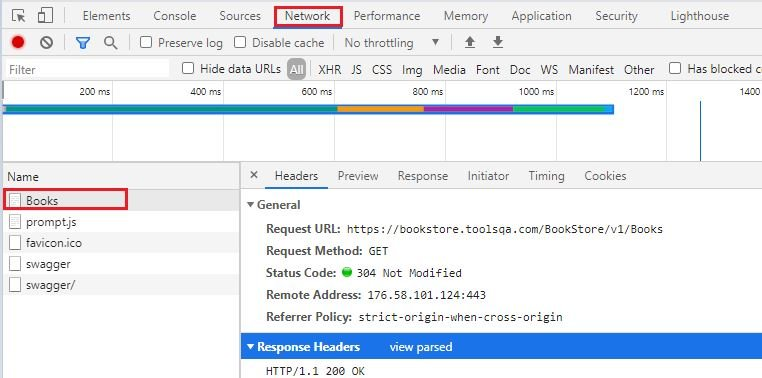
Ex: GET /BookStore/v1/Books HTTP/1.1



* 1. Zero or more headers



* 1. An optional request body
* HTTP Response
  1. Status Line

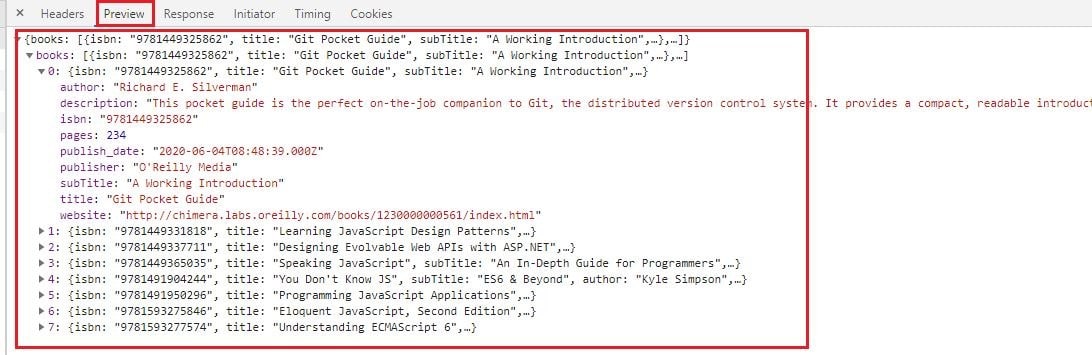


* 1. Zero or more headers

Graphical user interface, text, application

Description automatically generated

* 1. An optional message body



* Rest Architectural Elements:

| ***Element*** | ***Example*** |
| --- | --- |
| Resource | Information stored on a Server, which can be requested by a client. It could be Weather info or may be employee details |
| Resource Identifier | Now that we have a resource defined, we need to uniquely identify the resource. That is actually the complete URL |
| Representation | A resource is the actual data. Now this data can be represented as an XML, HTML or may be simple text. That is what is called a Representation |
| Representation Metadata | In order for the Clients to specify and process a resources given in a particular Representation (*XML or HTML etc*) some extra data (*Metadata*) needs to be passed in the request. |

A Resource can be

* Temporal
* Static

Temporal resource is one that keeps changing with time. For e.g. the Weather. A Static resource is one that stays constant over longer time durations. For e.g. a webpage containing some static text. Like the one you are reading now can just be a Static resource on the server.

**Authentication and Authorization in REST Web Services**

­Authentication: Is a process to prove that you are the person who you intend to be.

Basic Authentication Flow:

Taking the example of email login, we know that in order to Authenticate our self we have to provide a username and a Password. In a very basic Authentication flow using Username and Password, we will do the same thing in REST API call as well. but how do we send the Username and Password in the REST request?

A REST request can have a special header called **Authorization** Header, this header can contain the credentials (username and password) in some form. Once a request with Authorization Header is received, server can validate the credentials and can let you access the private resources.

In case of Authentication failures Server should respond with a status code of **401 Unauthorized.**

Authorization: Is the process of giving access to someone.

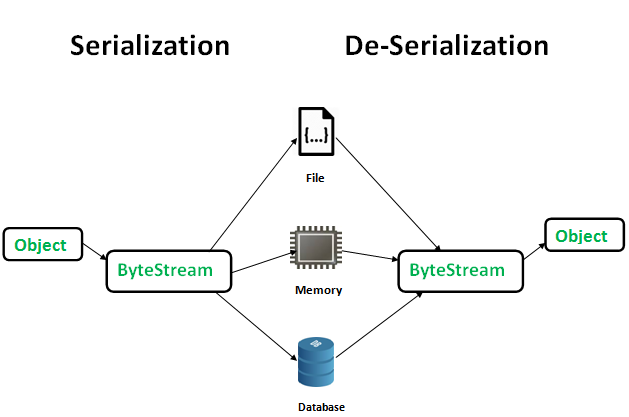
Involves checking resources that the user is authorized to access or modify via defined roles or claims. For example, the authenticated user is authorized for read access to a database but not allowed to modify it. The same can be applied to your API. Maybe most users can access certain resources or endpoints, but special admin users have privileged access.

Of course, these definitions are usually implemented together and interdependent, so when we refer to auth, we are referring to the overall system.

Authentication is stating that you are who are you are, and Authorization is asking if you have access to a certain resource.

**Serialization and Deserialization**

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.



Advantages of Serialization

1. To save/persist state of an object.
2. To travel an object across a network.

The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

To make a Java object serializable we implement the java.io.Serializable interface.

The ObjectOutputStream class contains writeObject() method for serializing an Object.

The ObjectInputStream class contains readObject() method for deserializing an object.

Only the objects of those classes can be serialized which are implementing java.io.Serializable interface.

Serializable is a marker interface (has no data member and method). It is used to “mark” java classes so that objects of these classes may get certain capability. Other examples of marker interfaces are:- Cloneable and Remote

**SerialVersionUID**

The Serialization runtime associates a version number with each Serializable class called a SerialVersionUID, which is used during Deserialization to verify that sender and receiver of a serialized object have loaded classes for that object which are compatible with respect to serialization. If the receiver has loaded a class for the object that has different UID than that of corresponding sender’s class, the Deserialization will result in an InvalidClassException. A Serializable class can declare its own UID explicitly by declaring a field name.

It must be static, final and of type long.

i.e- ANY-ACCESS-MODIFIER static final long serialVersionUID=42L;

If a serializable class doesn’t explicitly declare a serialVersionUID, then the serialization runtime will calculate a default one for that class based on various aspects of class, as described in Java Object Serialization Specification. However it is strongly recommended that all serializable classes explicitly declare serialVersionUID value, since its computation is highly sensitive to class details that may vary depending on compiler implementations, any change in class or using different id may affect the serialized data.

It is also recommended to use private modifier for UID since it is not useful as inherited member.