REST Assured - Advanced

Authentication & Authorization, API Chaining and building Framework

Authentication & Authorization Types

Authentication vs Authorization

Authentication is to find out who you are, authorization is to find out what you can do.

Authentication Types

- Basic Authentication: (username and password)
- Session based Authentication: (SessionID)
- Token based Authentication: (Token)
- API Key

Authorization Type

- OAuth 2.0

Session & Cookie

Session:

It represents the stateful connection established between the server and the client (usually a web browser) during the user's visit to a website. Every session is uniquely identified by **SESSION_ID**.

Cookies:

A cookie is a small piece of data that a website can store on a user's browser. It is typically used to store key-value pairs that persist in the Request Headers. (Every website has their own cookies)

Session Based Authentication

Step 1: User submit correct credentials (username and password) to the backend server

Step 2: Backend server establish a session with its SESSION_ID, and send back a response that set the SESSION_ID in the cookie (there's a field called SET_COOKIE in the response header)

Step 3: Browser stored the cookie after receiving the response

Step 4: In the following requests sent by that user, all the request headers will contain the cookies.

Step 5: Backend server will retrieve SESSION_ID from cookies to identify the user.

Token Based Authentication

One important principle of RESTful API is Stateless, so session becomes unsuitable for our RESTful API. Instead we need to use token for authentication.

JWT Token: https://jwt.io/

Step 1: User submit correct credentials (username and password) to the backend server

Step 2: Backend server send back a Jwt Token (which contains necessary user information) and stored in the client browser's cookie (or in the)

Step 3: For all the following requests sent by that user, the backend server will parse the Jwt Token attached in the cookie to identify the user.

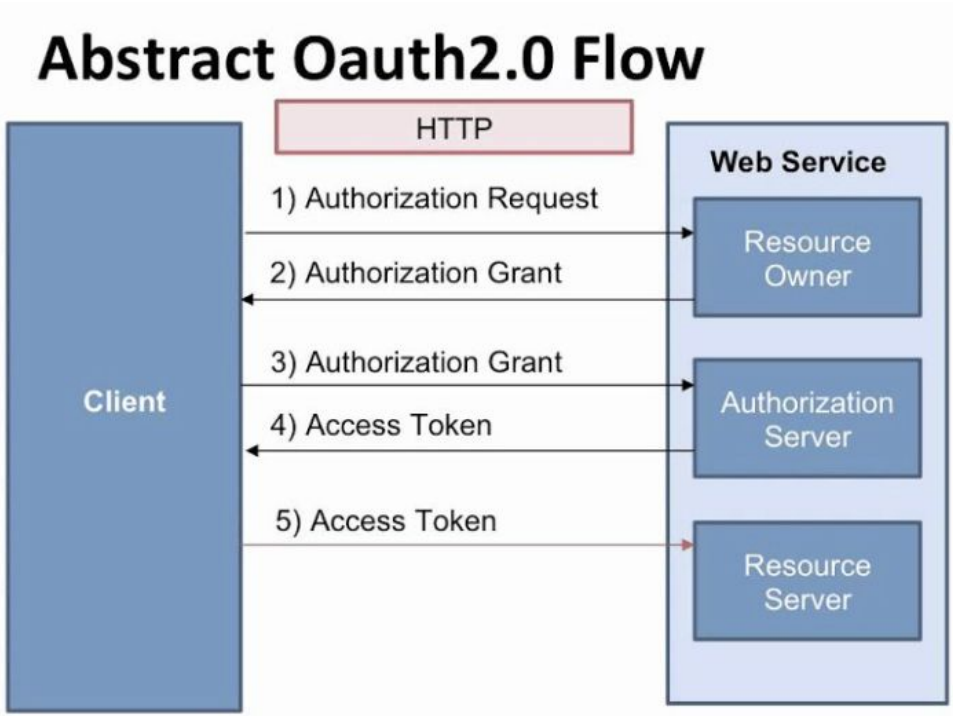
OAuth 2.0

What is OAuth 2.0?

OAuth 2.0 is an authorization framework that allows users (**resource owner**) to grant limited access to their resources on one website (known as the "**resource server**") to another website or application (known as the "**client**").

Real World Scenario:

There's an online photo printing service, where users can upload their photos through Google Photos Service. If you want to print your photos in the Google Photos, then the photos are called resources, you are the **Resource Owner,** Google Photos are the **Resource Server,** and the online photo printing service is the **Client Application.** Of course you don't want the Client Application to have full access to your photos, so you need to grant **limited access** (for example a folder).



Attentions:

- Client application must register with authorization server to get client id and secret.
- Once the resource owner grant the authorization, the authorization server will return an authorization code, which can only be used once to exchange for access token (for security) !

API Chaining

Scenario: I want to perform an end-to-end API Testing, and here're my steps:

- Create a resource, (usually receive an Id in the response)
- Get the resource info by Id
- Update the resource by Id
- Delete the resource by Id

Question: How do I pass the ID generated at the first step to the following steps?

Challenge: How do I pass the data between the test methods across different classes?

Building a framework of API Testing

Principles:

- **Reusability:**
  - How to reduce repeated code?
    - Request/Response Specifications
- **Scalability:**
  - How to make sure your framework still performs efficiently when there are hundreds of endpoints to test?
    - Parallel Execution
    - Request/Response Compression (Pagination, Partially Update, etc)
- **Resource Management:**
  - How to manage hundreds of endpoints, tokens, test data and so on.
    - Resource file in yaml, json format
    - Data Providers