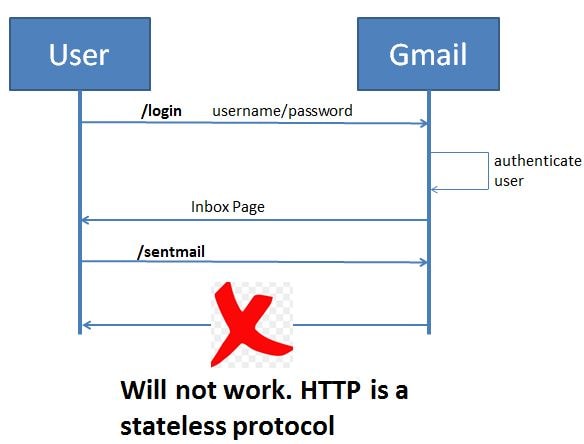
**Understanding the need for JSON Web Token(JWT)**

* JWT stands for JSON Web Token
* It is pronounced as JAWT
* It is Open Standard - RFC7519
* JWT makes it possible to communicate securely between two bodies
* JWT is used for Authorization

Let us first understand what is Authorization and need for it.

In order to understand Authorization we will be taking example of user interaction with Gmail. Consider a scenario where a user wants to access his Gmail inbox page. This will involve user interaction with the Gmail server. For this the user will be sending HTTP requests to Gmail server and in response will expect the response from Gmail Server.  
  


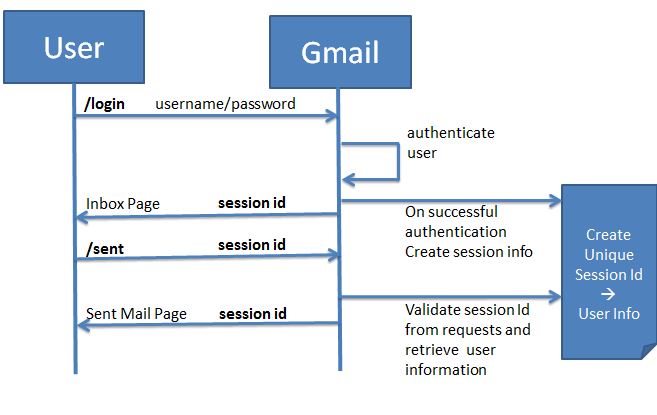
The steps will be as follows-

* The user will send a http request to Gmail server with url /login. Along with this request the user will also be sending the username and password for authentication.
* The Gmail server will authenticate this request if it is successful it will return the Gmail inbox page as response to the user.
* Now suppose the user wants to access his sent mail page, so he will again send a request to the Gmail server with url /sentmails. This time he will not be sending the username and password since he user has already auntenticated himself in the first request.
* The user expects Gmail to return the sent mail page. However this will not be the case. **The Gmail server will not return the sent mail page but will instead not recognize the user.**

The reason for this is HTTP is a stateless protocol. As the name suggests no state is maintained by HTTP. So in case of HTTP protocol for the Gmail server each request is from a new user. It cannot distinguish between new and returning users. One solution for this issue could be to pass username and password along with each request. But then this is not a good solution.  
**Once a user has been authenticated. For all subsequent requests the user should be authorized to perform allowed operations**  
Authorization can be implemented using

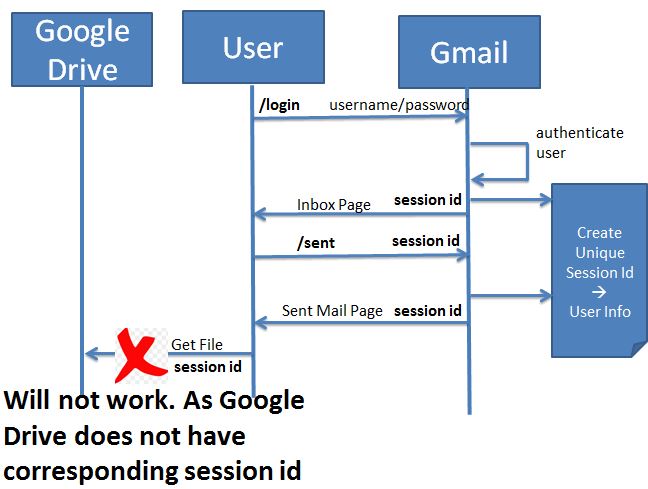
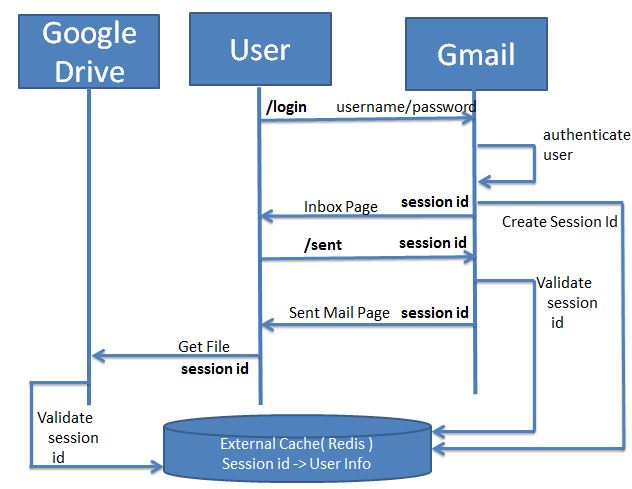
* Session Management
* JSON Web Token

Using Session Management for Authorization

In case of Session Management, once the user has been authenticated then the Gmail Server will create a unique session Id. Corresponding to this session id it will store in memory all the user information that is needed by the Gmail server for recognizing the user and allowing it perform operations.  
Also then for all subsequent requests and response, this session id will also be passed. So now when the server receives a request it will check the session id. Using this session id will check if there is any corresponding information. It will then allow the user to access the resource and return back the response along with the session id.  


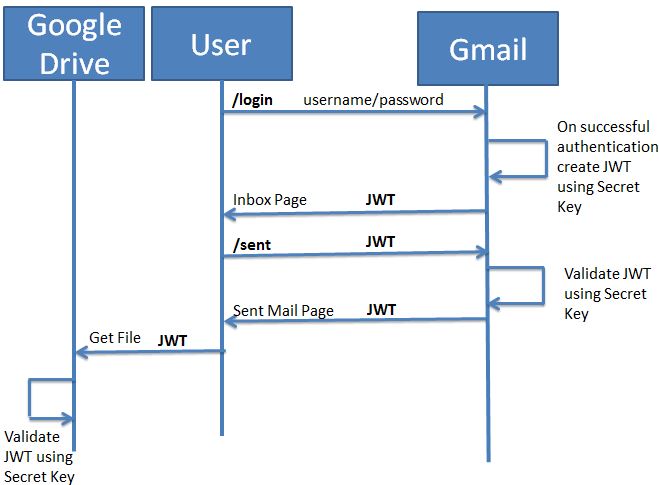
Drawbacks of Session Management for Authorization

* Session Id is not self contained. It is a reference token. During each validation the Gmail server needs to fetch the information corresponding to it.
* Not suitable for microservices architecture involving multiple API's and servers

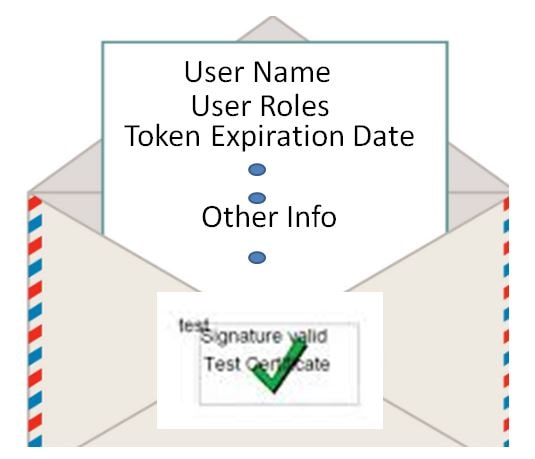
  
  


Using JWT for Authorization

We make use of the JWT for authorization so the server will know that the user is already authenticated and so the user does not need to send the credentials with each and every request.



Advantages of JWT Authorization



* JWT is self contained. It is a value token. So during each validation the Gmail server does not needs to fetch the information corresponding to it.
* It is digitally signed so if any one modifies it the server will know about it
* It is most suitable for Microservices Architecture
* It has other advantages like specifying the expiration time.