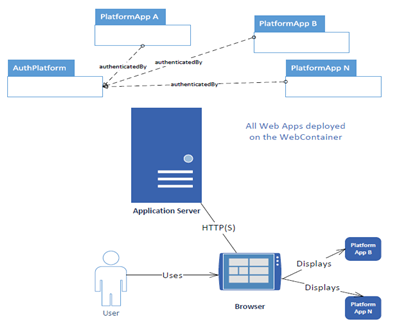
**My own primitive Single Sign On**

I have often worked on web applications which used **SSO**. In fact my blogger profile also works through an SSO flow. If I am logged in into Gmail than all fine, else I need to login into Gmail or more specifically my Google account.  
One way to have an SSO flow is :

1. A **user** attempts to access a **resource**.
2. The **resource provider** checks if the user is authenticated.
3. If not he is redirected to an **Identity Provider** whose job it is to authenticate the user. The Provider may also do the authorization checks or he may let the resource provider handle the same.
4. Once the above process completes - or the **identity provider communicates user details to the resource provider**, the resource provider than takes action on the user request.

The important thing here is the **communication mechanism** used between the identity provider and the resource provider. In one project my team achieved the SSO using **SAML over HTTPS**.  
To be entirely honest I have no knowledge of SAML or how it works. But I still wanted to get SSO done. I decided to build my own web platform to provide SSO to associated web apps that acted as resource providers.  
Since I am trying to keep things simple, I am gonna stay from terms like federation server,  active directory etc, SAML. In my development, the only thing that exists are Servlets and JSP.  
I decided to do some designing and this is a rough diagram of my whole platform:

[[](http://2.bp.blogspot.com/-k1eCfBbNgRo/UyQMFVPLJ0I/AAAAAAAAAsc/UnpYOq4MSuI/s1600/AuthPlatform.png)](http://2.bp.blogspot.com/-k1eCfBbNgRo/UyQMFVPLJ0I/AAAAAAAAAsc/UnpYOq4MSuI/s1600/AuthPlatform.png)

Here we have a web app called the AuthPlatform that is responsible for performing authentication of all users trying to use the resources exposed by PlatformAppX where X is A,B,.....N (Sounds like a algebra eqn, but no, I'm not going anywhere near Maths here).  
So how does the solution work?  
When the u**ser tries to get at a resource** exposed by a PlatformApp (say PlatformAppA).

1. PlatformAppA checks if a user session exists.
2. If not, than PlatformAppA will ask the AuthPlatform to authenticate the user.
3. Once AuthPlatform completes authentication, it will redirect user to PlatformAppA.

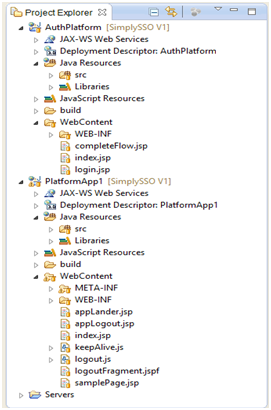
When **authenticated user attempts to use another resource**(say PlatformAppB)

1. PlatformAppB checks with AuthPlatform to verify if user is autenticated.
2. Once it verifies the authentication, PlatformAppB allows the application to proceed.

When **user decides to logout** from the application

1. The application informs the AuthPlatform about user's decision to terminate the session. He is also logged out from the AuthPlatform. (This is similar to logging out from Gmail.)
2. The user is logged out from the Platform App. If he tries to perform any operations in any other PlatformAPP he should see a "session does not exist, Please sign in" message.

Accordingly I created the projects in Eclipse:

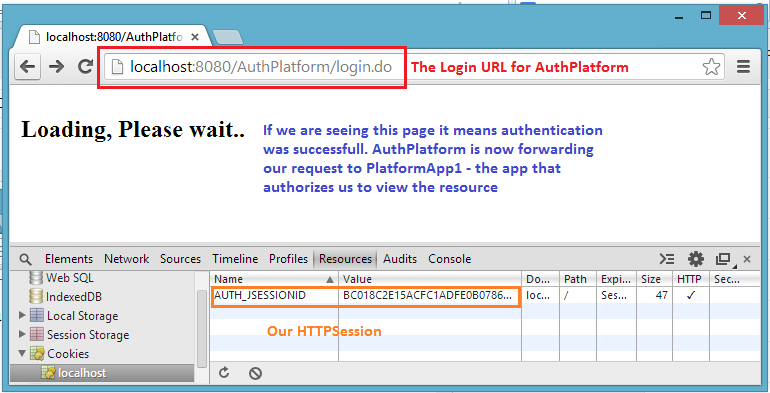
[](http://4.bp.blogspot.com/-y9xsyX06YOY/UyQPK28xYLI/AAAAAAAAAsw/FWBVGvDL-ag/s1600/ProjectStructure.png)

**My own primitive Single Sign On - Part 2**

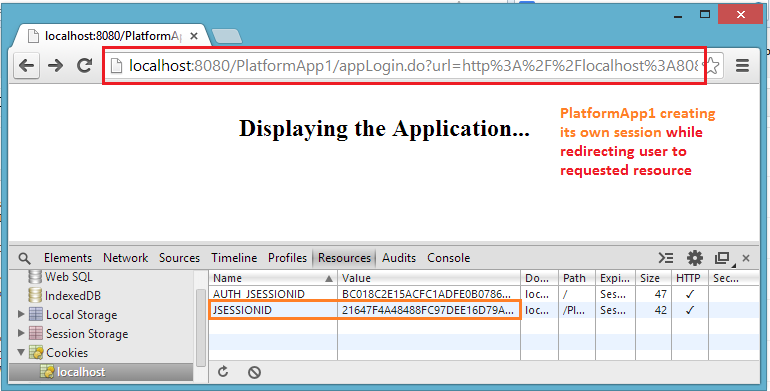
In the previous post I started on [creating of my custom SSO solution](http://learningviacode.blogspot.com/2014/03/my-own-primitive-single-sign-on.html). I have decided to name it "SimplySSO". To continue with the application flow, lets start with the login flow:  
User attempts to access PlatformAppA:

[](http://2.bp.blogspot.com/-3eMRqHfH0lg/UyQSZS4A-3I/AAAAAAAAAtQ/v0I4t5GCYGE/s1600/index.jsp.png)

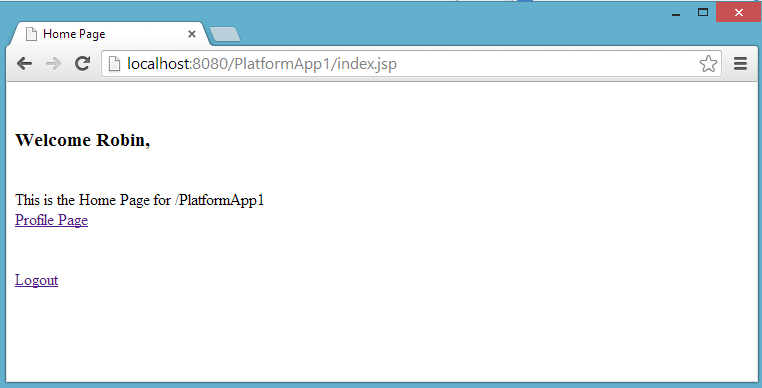
 Since the user was not logged in, he was redirected to the AuthPlatform. The AuthPlatform immediately responded with a login page. Once the user submitted his credentials, the AuthPlatform began the process of authentication:  
Once the AuthPlatform has authenticated the user, it will create a session (yes HTTP Session) to track this user's requests. It will now redirect the user back to requested resource - PlatformApp1/index.jsp

[](http://3.bp.blogspot.com/-U_8EiETBIFw/UyQVGHKtUqI/AAAAAAAAAtk/aRqGBf6h2lg/s1600/login.do.png)

PlatformApp1 will also have some infomration that it would like to maintain in a user session. It cannot use the AuthPlatform session as that belongs to the AuthPlatform. To avoid pollution of AuthPlatform session, we have each PlatformApp create its own session.

[](http://2.bp.blogspot.com/-7AXe3MgQMHQ/UyQWDWtaItI/AAAAAAAAAtw/XI2RG5J2WUg/s1600/appLogin.do.png)

 Now that all setup is complete, the user can see his page (finally !)

[](http://3.bp.blogspot.com/-pllaTnLYMmM/UyQWglX0i_I/AAAAAAAAAt4/yHstpJCuU2k/s1600/app1_index.jsp.png)

**A question that comes here is how did the PlatformApp find out about the existence of an authenticated session for user Robin ?** There is a dirty trick here that I haven't spoken about yet. Lets look at the session information again:

[](http://2.bp.blogspot.com/-U8ete8Vw65w/UyQanbMiq2I/AAAAAAAAAuE/gz1Wc12uyfg/s1600/sessions.png)

Now that we have seen how this is done for one PlatformApp, lets try and [extend this behavior to N apps](http://learningviacode.blogspot.com/2014/03/my-own-primitive-single-sign-on-last-lap.html).

**My own primitive Single Sign On - Last Lap**

In our previous post we saw how [PlatformApp1 was able to read the cookie created by AuthPlatform](http://learningviacode.blogspot.com/2014/03/my-own-primitive-single-sign-on-part-2.html). The PlatformApp1 verifies with the AuthPlatform if this cookie is valid (maybe through shared data or a simple REST API exposed by AuthPlatform) and accordingly lets the user proceed.  
Now we have a authenticated user who decides he wants to use PlatformApp2. The same logic works.  
Here too when the first request was made for a page:

[](http://2.bp.blogspot.com/-d6YZtU8zSfA/UyQeBVqd9kI/AAAAAAAAAuQ/menjqoaQVD8/s1600/PlatformApp2.png)

as seen here the browser sent details of the root level cookie (The one that belongs to AuthPlatform).  
PlatformApp2 verifies if this session is valid and is true allows the user to view the page.  
And..........  
Thus we have Single Sign On.  
  
**What about Logout ??**  
Well for logout from any application, we need to void both sessions  - the PlatformApp from which user asked to logout and also the AuthPlatform.  
With the AuthPlatform session gone, now no PlatformApp will work. (Just like in Google - sign out of one application(say Gmail) means you are signed out of all apps).  
(I used a little AJAX to get this done- not my best use of AJAX)  
  
**What about inactive sessions ?**  
I used the standard approach to keeping sessions alive. A keep alive hit to both servers every few minutes to avoid inactive timeout. AJAX again.  
The AuthPlatform cookie exists  for a long time, the PlatformApp cookies are invalidated once the browser window/tab is closed.  
  
Thus we were able to achieve SSO using just JSP and Servlets. Nothing complex. No third party tools etc. Obviously there will be shortcomings to this primitive approach. The most obvious one being the cookie and path fiddling needs to be done.To make the communication between web apps easy I used cross context access - means all webapps need to be in the same context. (Of course this issue can be easily sorted out).  
I have uploaded the code for the above work on my GIT account - <https://github.com/RobinVJ/SimplySSO/> (Branch V1)  
Feel free to download and test out. All comments and feedback are welcome - as always.