**Mutual Authentication**

With *mutual authentication*, the server and the client authenticate each other. Mutual authentication is of two types:

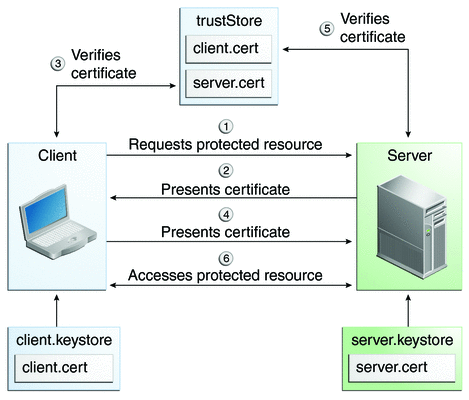
* Certificate-based (see [Figure 25–4](https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpkl/index.html#bncbu))
* User name/password-based (see [Figure 25–5](https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpkl/index.html#bncbv))

When using certificate-based mutual authentication, the following actions occur.

1. A client requests access to a protected resource.
2. The web server presents its certificate to the client.
3. The client verifies the server’s certificate.
4. If successful, the client sends its certificate to the server.
5. The server verifies the client’s credentials.
6. If successful, the server grants access to the protected resource requested by the client.

[Figure 25–4](https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpkl/index.html#bncbu) shows what occurs during certificate-based mutual authentication.

**Figure 25–4 Certificate-Based Mutual Authentication**



In user name/password-based mutual authentication, the following actions occur.

1. A client requests access to a protected resource.
2. The web server presents its certificate to the client.
3. The client verifies the server’s certificate.
4. If successful, the client sends its user name and password to the server, which verifies the client’s credentials.
5. If the verification is successful, the server grants access to the protected resource requested by the client.

[Figure 25–5](https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cpkl/index.html#bncbv) shows what occurs during user name/password-based mutual authentication.

**Figure 25–5 User Name/Password-Based Mutual Authentication**

Diagram of five steps in mutual authentication with user
name and password  
  
**Session in Java**

The time interval in which two systems (i.e. the client and the server) communicate with each other can be termed as a session. In simpler terms, a session is a state consisting of several requests and response between the client and the server.

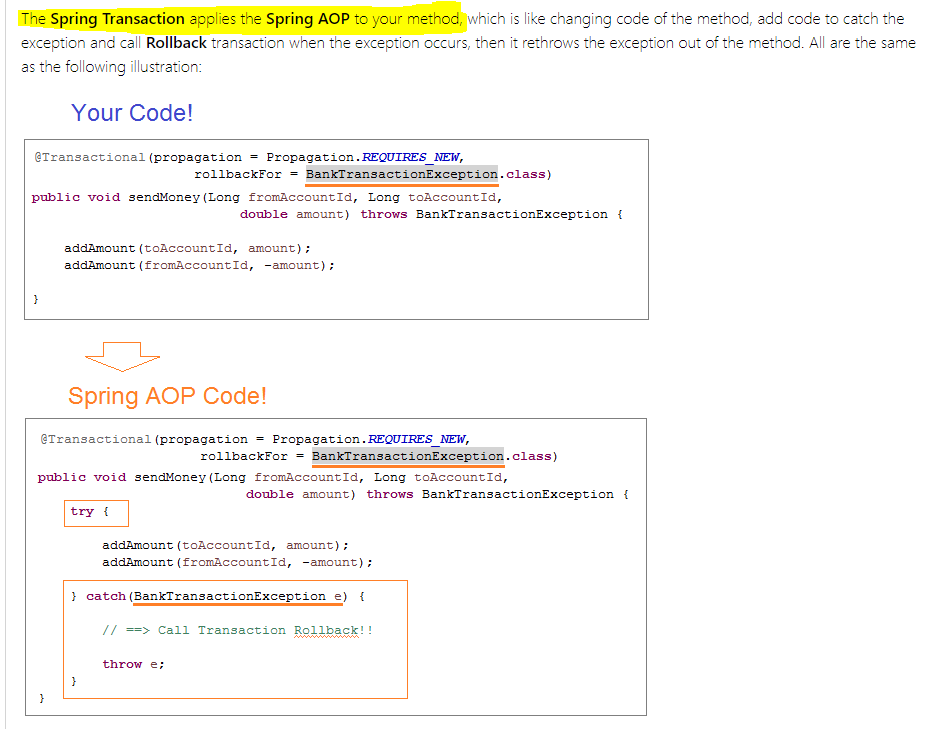
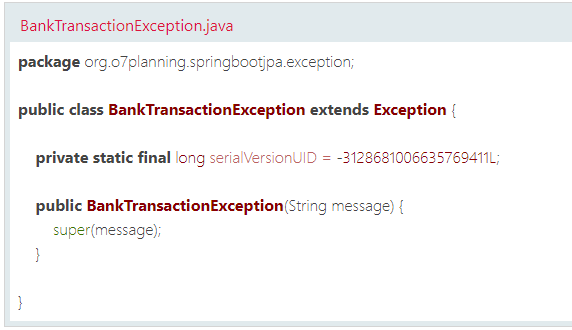
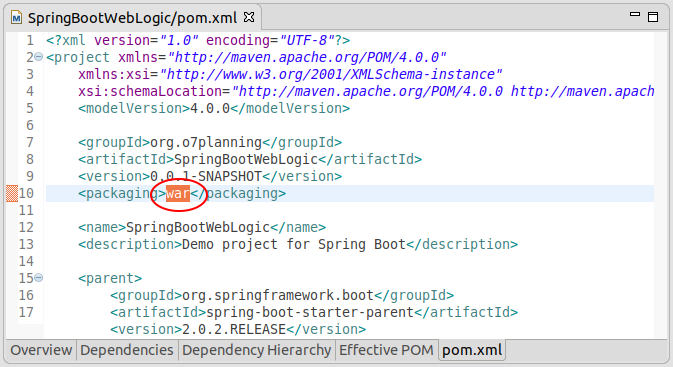
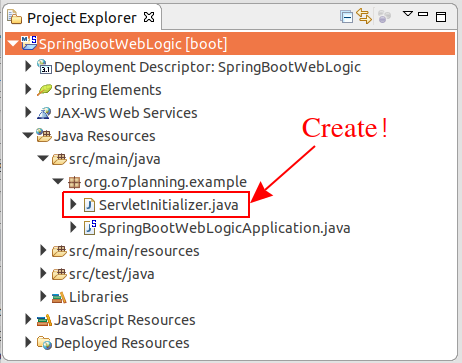
It is a known fact that HTTP and Web Servers are both stateless. Hence, the only way to maintain the state of the user is by making use of technologies that implement session tracking.  
Session tracking in servlets can be implemented by a number of methods, cookies being one of them. However, they have   
  
multiple disadvantages:

Only textual information can be kept by them.

If cookies are disabled by a user, the web application is unable to make use of them.

Not more than 4kb of data can be contained by a single cookie.

Another way to implement session tracking is by creating sessions with unique session ids for every user in a java servlet.

<https://o7planning.org/11661/spring-boot-jpa-and-spring-transaction>  
  
  
  
  
  
  
**How to save data once only even if we request save() multiple times.**The difference between POST and PUT is that PUT requests are idempotent. That is, ***calling the same PUT request multiple times will always produce the same result***. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.  
  
<https://www.w3schools.com/tags/ref_httpmethods.asp>  
  
**Deploy Spring Boot Application on Oracle WebLogic Server**<https://o7planning.org/11901/deploy-spring-boot-application-on-oracle-weblogic-server>1) In pom.xml file we need to change packaging as ***war***.  
  
  
2) create ServletInitializer class  


3) ServletInitializer.java

**package** org.o7planning.example;

**import** org.springframework.boot.builder.SpringApplicationBuilder;

**import** org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

**public** **class** **ServletInitializer** **extends** **SpringBootServletInitializer** {

@Override

**protected** SpringApplicationBuilder **configure**(SpringApplicationBuilder application) {

**return** application.sources(SpringBootWebLogicApplication.class);

}

}

\*\* pom.xml \*\*

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-tomcat</**artifactId**>

<**scope**>provided</**scope**>

</**dependency**>

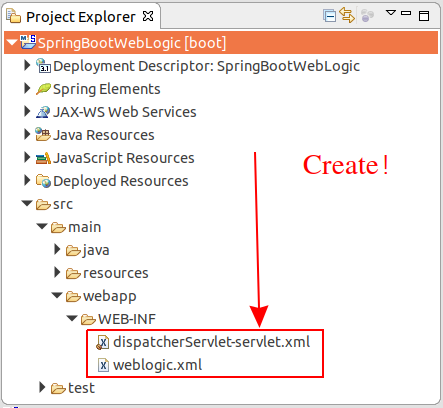
***If there are more than one Application class in your project, please tell the Spring Boot which class will be used for your application.***

***\*\* pom.xml \*\****

<properties>

<start-class>org.o7planning.example.OtherSpringBootWebApplication</start-class>

</properties>  
 **weblogic.xml & dispatcherServlet-servlet.xml**

Create the 2 files such as weblogic.xml & dispatcherServlet-servlet.xml in the src/main/webapp/WEB-INF folder. It is noted that if this folder doesn't exist, please create it.  
  


When your application is deployed on the **WebLogic,**there will be **"context-root = /myweb"**, and you can change it by installing in the **weblogic.xml**file:

**weblogic.xml**

<?xml version="1.0" encoding="UTF-8"?>

<wls:weblogic-web-app

xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-web-app

http://xmlns.oracle.com/weblogic/weblogic-web-app/1.4/weblogic-web-app.xsd">

<wls:context-root>/myweb</wls:context-root>

<wls:container-descriptor>

<wls:prefer-application-packages>

<wls:package-name>org.slf4j.\*</wls:package-name>

<wls:package-name>org.springframework.\*</wls:package-name>

</wls:prefer-application-packages>

</wls:container-descriptor>

</wls:weblogic-web-app>

**dispatcherServlet-servlet.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

</beans>