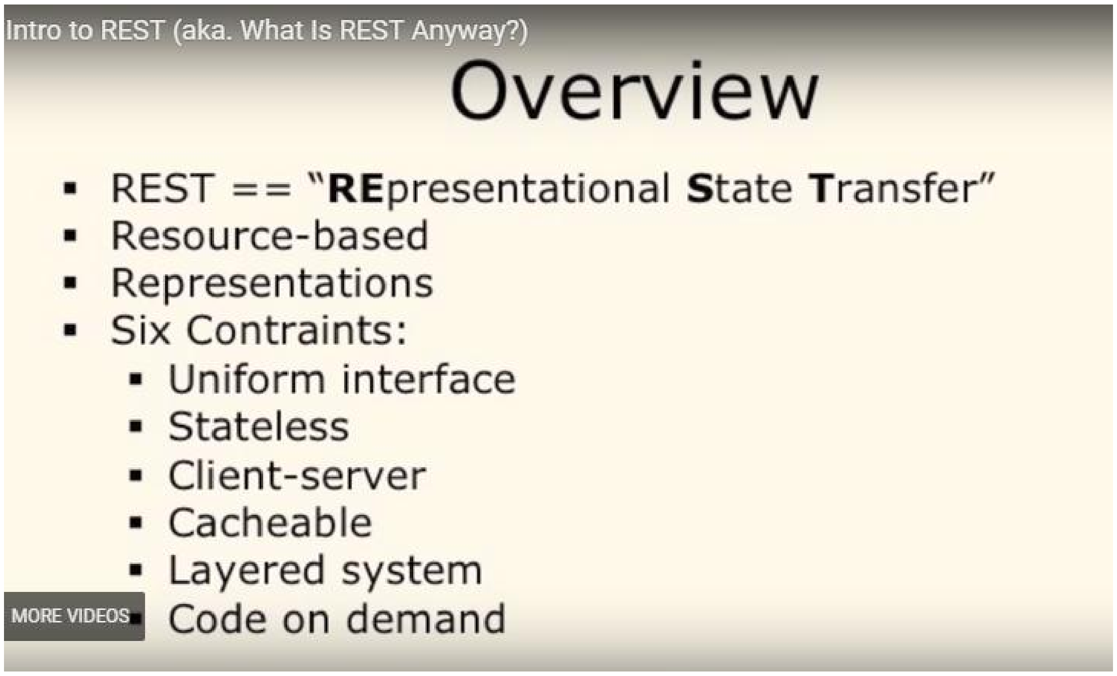
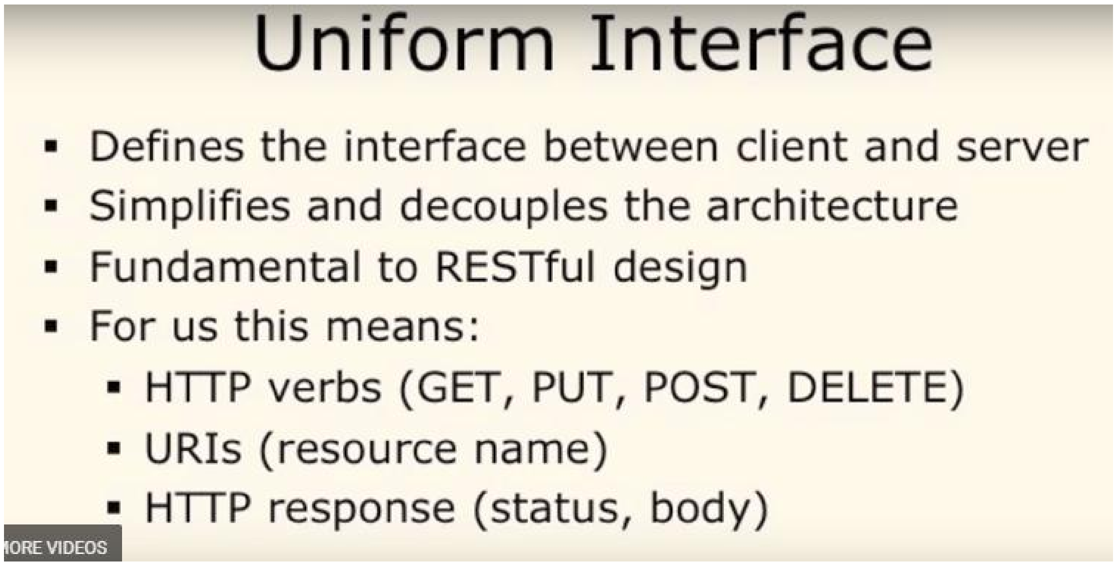

Intro to REST (aka. What Is REST Anyway?)

Overview

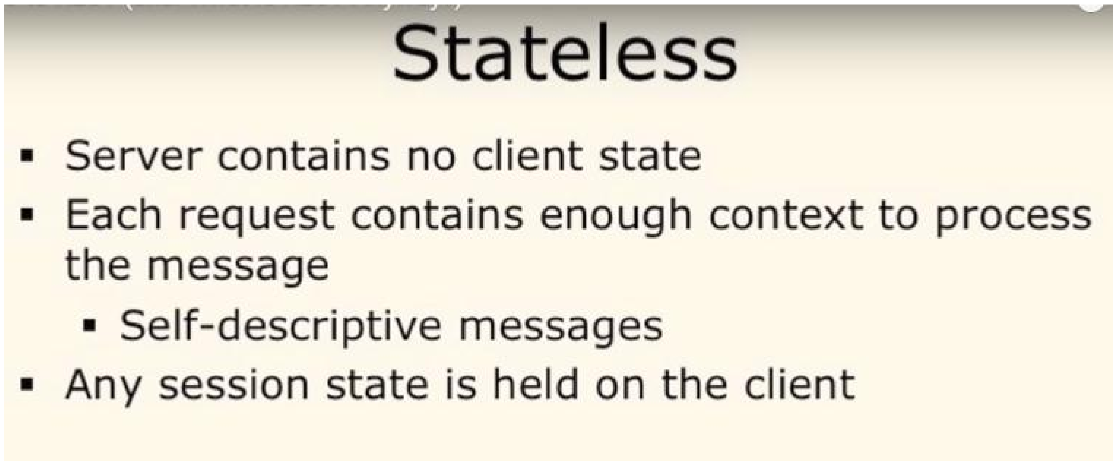
- REST == "**RE**presentational **S**tate **T**ransfer"
- Resource-based
- Representations
- Six Contraints:
  - Uniform interface
  - Stateless
  - Client-server
  - Cacheable
  - Layered system
  - Code on demand

Resource based, resources are identified by some URI.



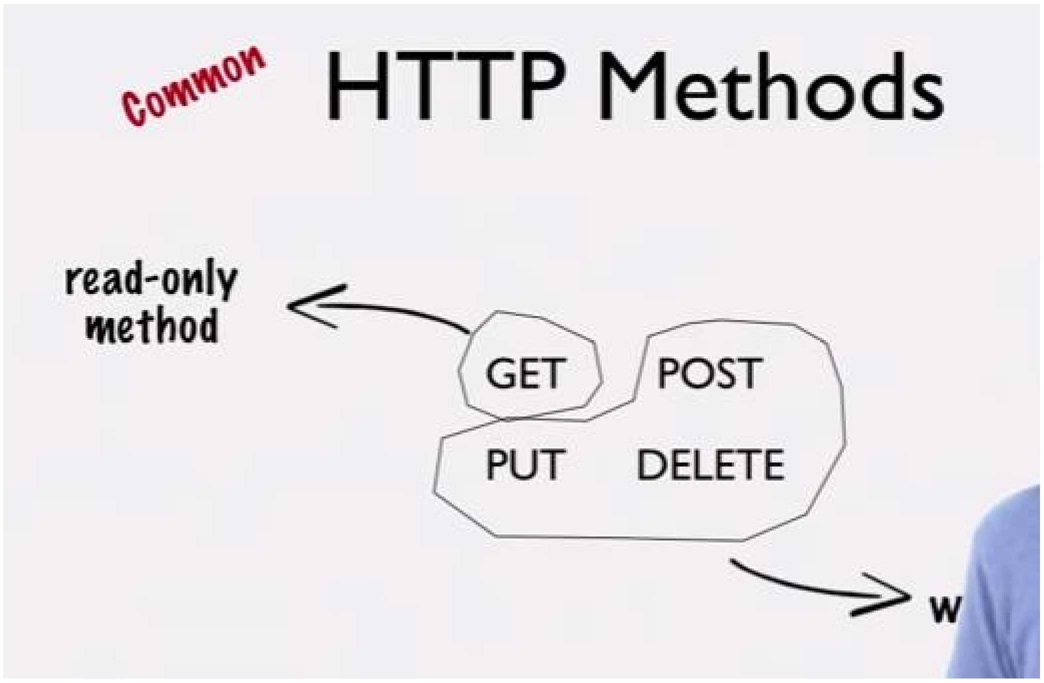
Uniform Interface

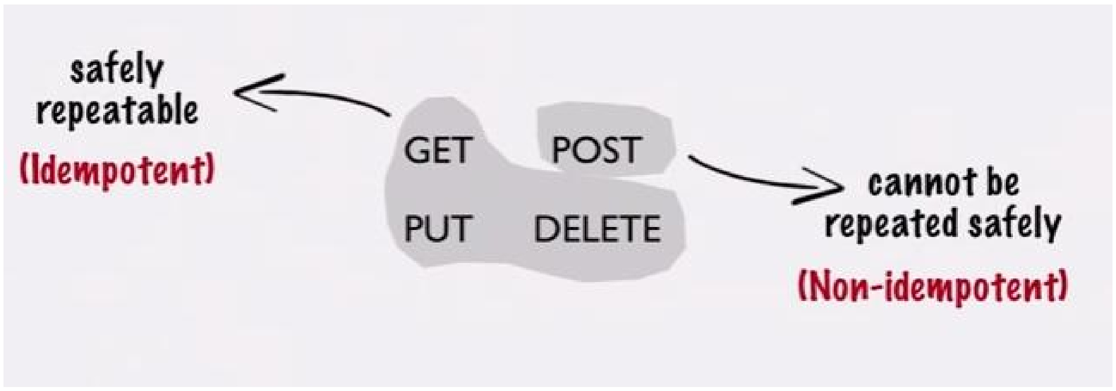
- Defines the interface between client and server
- Simplifies and decouples the architecture
- Fundamental to RESTful design
- For us this means:
  - HTTP verbs (GET, PUT, POST, DELETE)
  - URIs (resource name)
  - HTTP response (status, body)

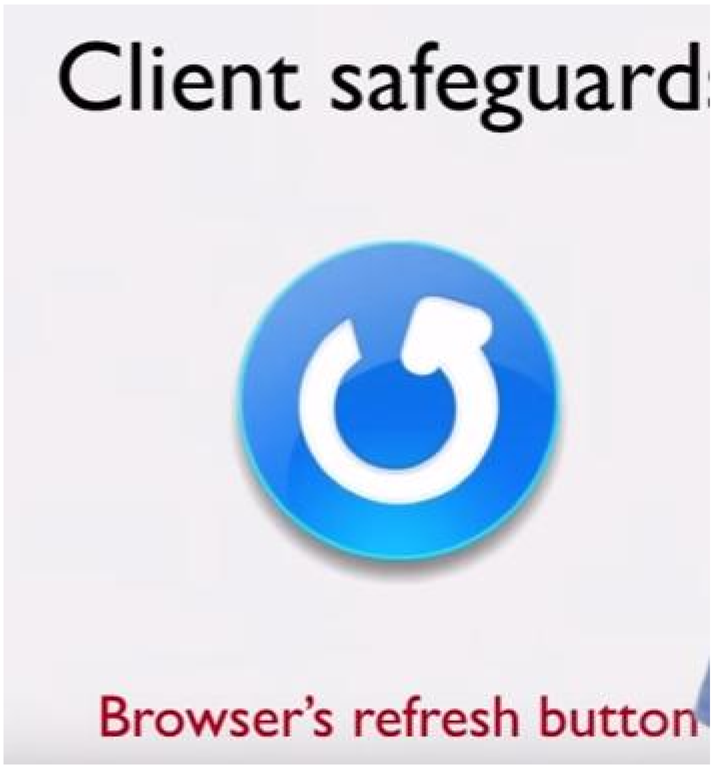


Stateless

- Server contains no client state
- Each request contains enough context to process the message
  - Self-descriptive messages
- Any session state is held on the client





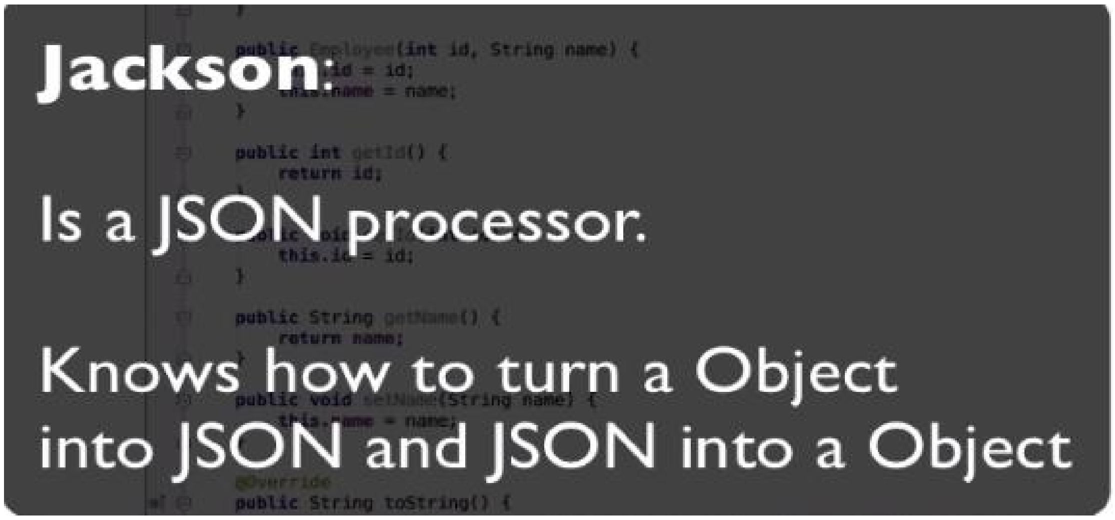


Client safeguard

Browser's refresh button

@JsonAutoDetect

on top of pojo class

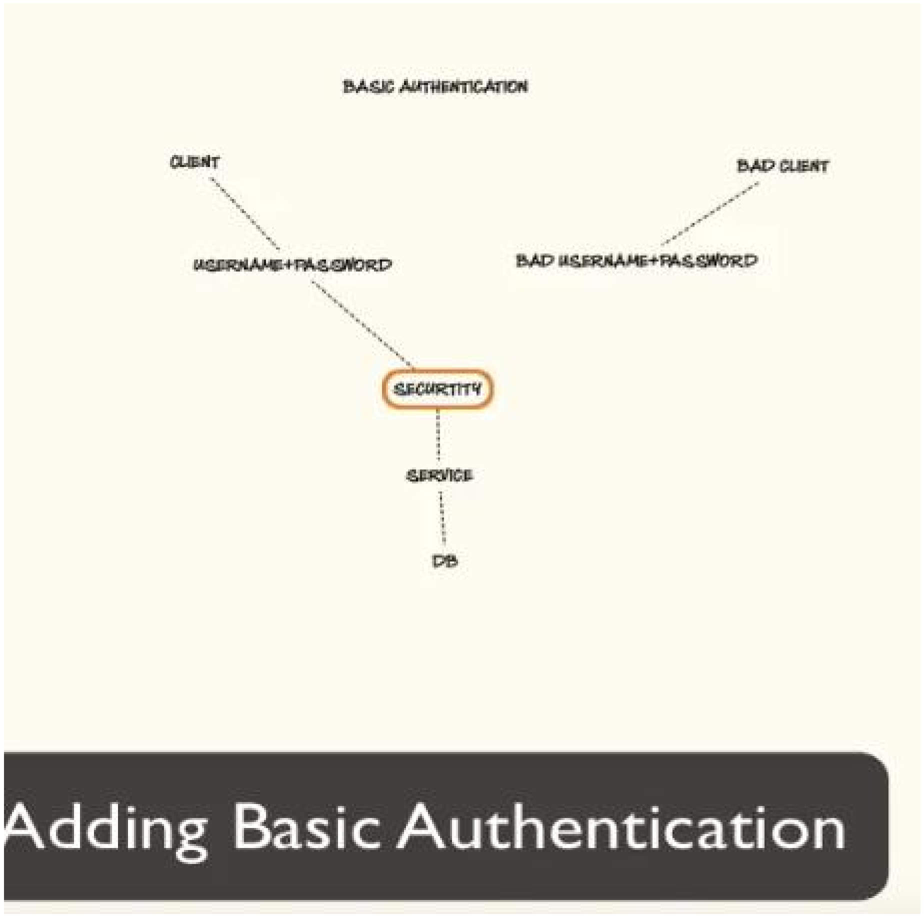


Jackson:

Is a JSON processor.

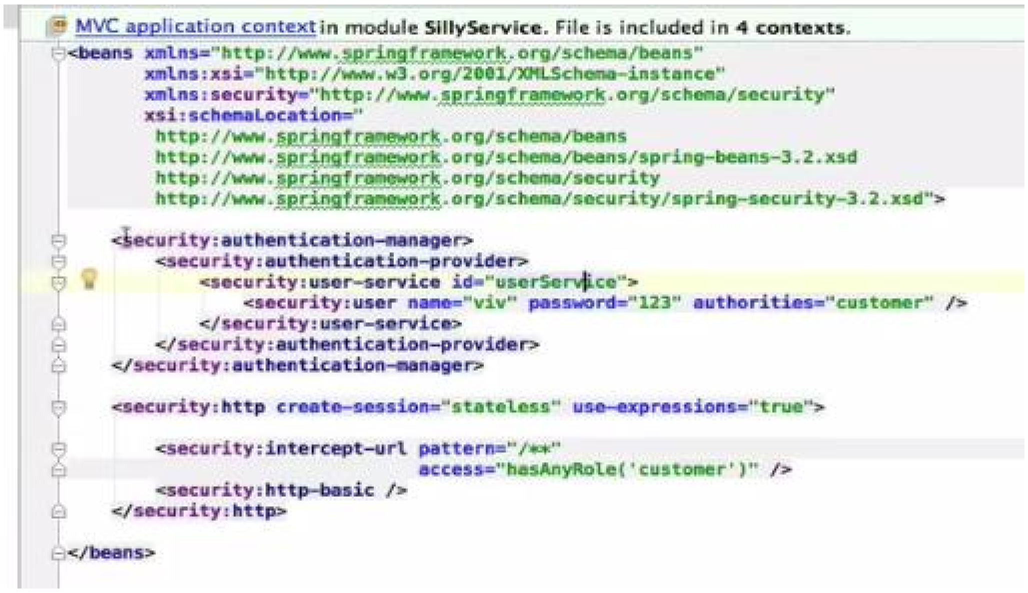
Knows how to turn a Object into JSON and JSON into a Object

Securing our Spring REST service



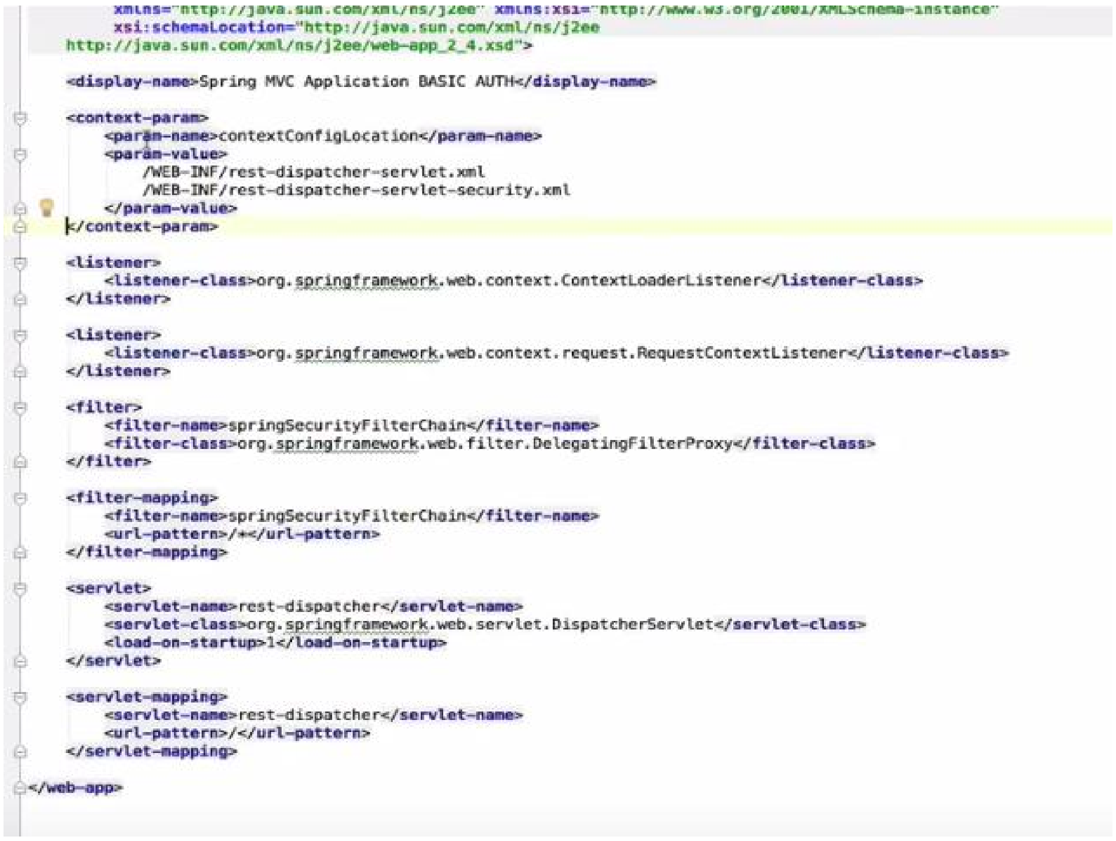
Adding Basic Authentication

Basic spring security authentication manager and intercepter

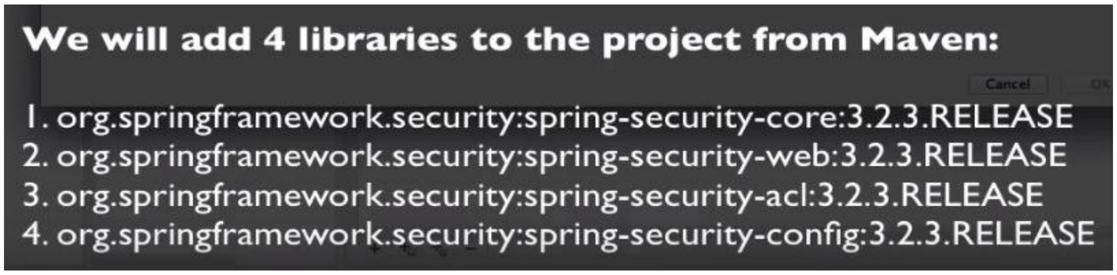


MVC application context in module **SillyService**. File is included in **4 contexts**.

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security-3.2.xsd">

    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service id="userService">
                <security:user name="viv" password="123" authorities="customer" />
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>

    <security:http create-session="stateless" use-expressions="true">

        <security:intercept-url pattern="/**"
                                access="hasAnyRole('customer')" />
        <security:http-basic />
    </security:http>

</beans>
```



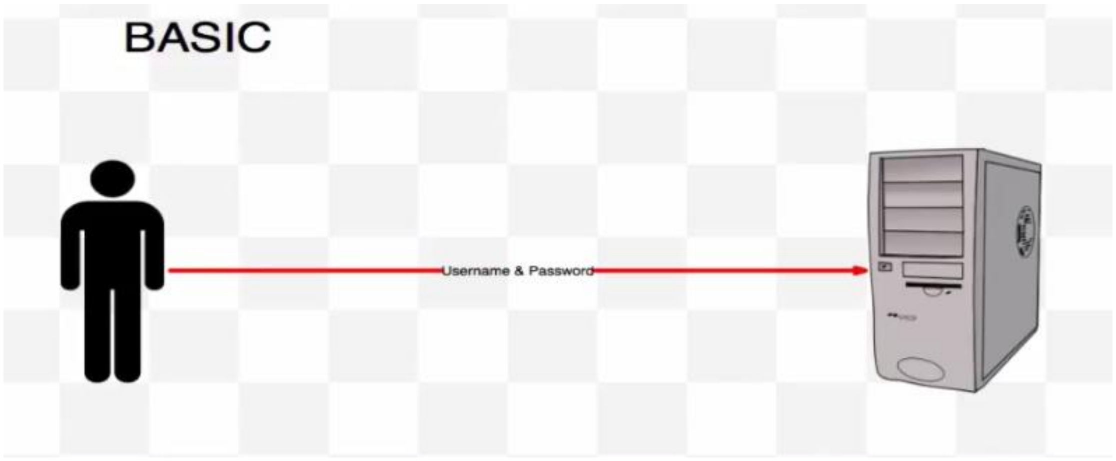
```xml
         xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Spring MVC Application BASIC AUTH</display-name>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/rest-dispatcher-servlet.xml
            /WEB-INF/rest-dispatcher-servlet-security.xml
        </param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <listener>
        <listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>
    </listener>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>rest-dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>rest-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```



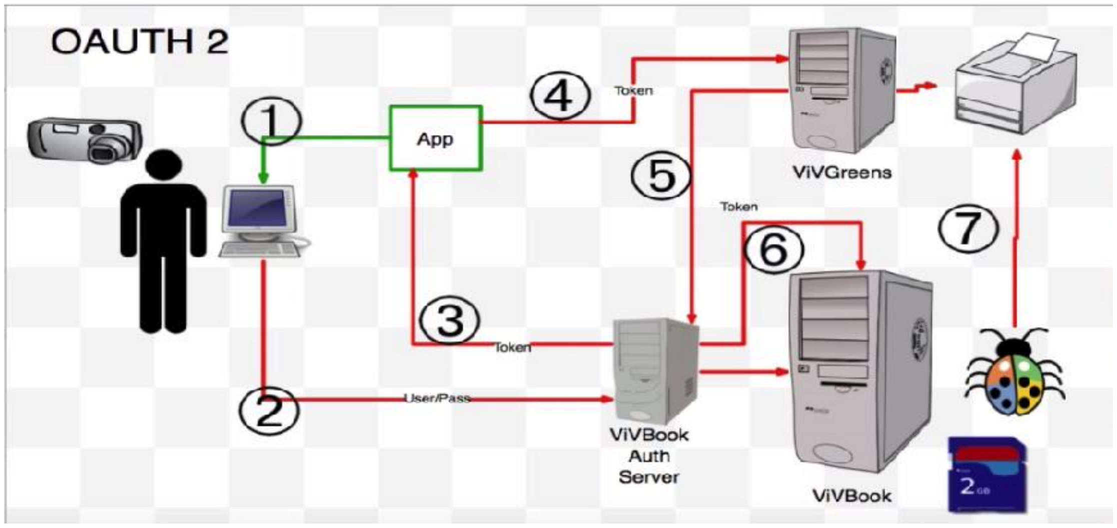
We will add 4 libraries to the project from Maven:

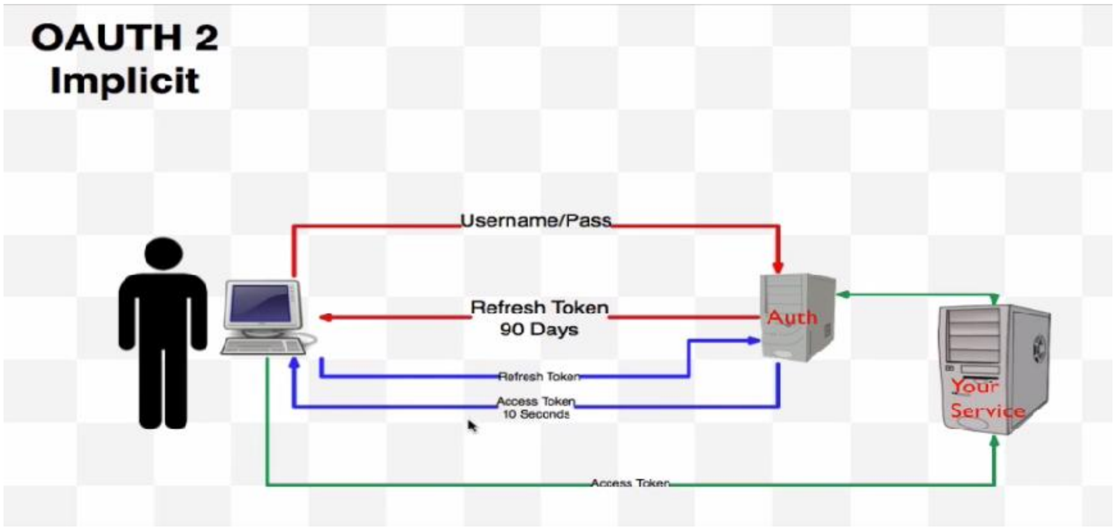
1. org.springframework.security:spring-security-core:3.2.3.RELEASE
2. org.springframework.security:spring-security-web:3.2.3.RELEASE
3. org.springframework.security:spring-security-acl:3.2.3.RELEASE
4. org.springframework.security:spring-security-config:3.2.3.RELEASE

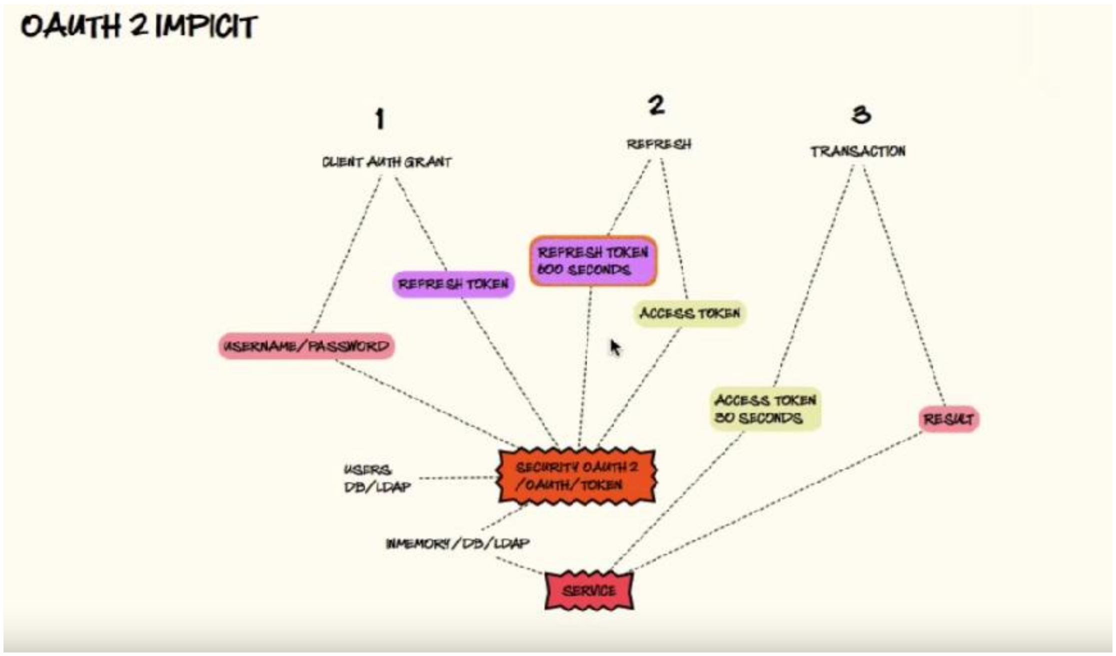
Username/Password, Base 64 encoded and sent through hader.



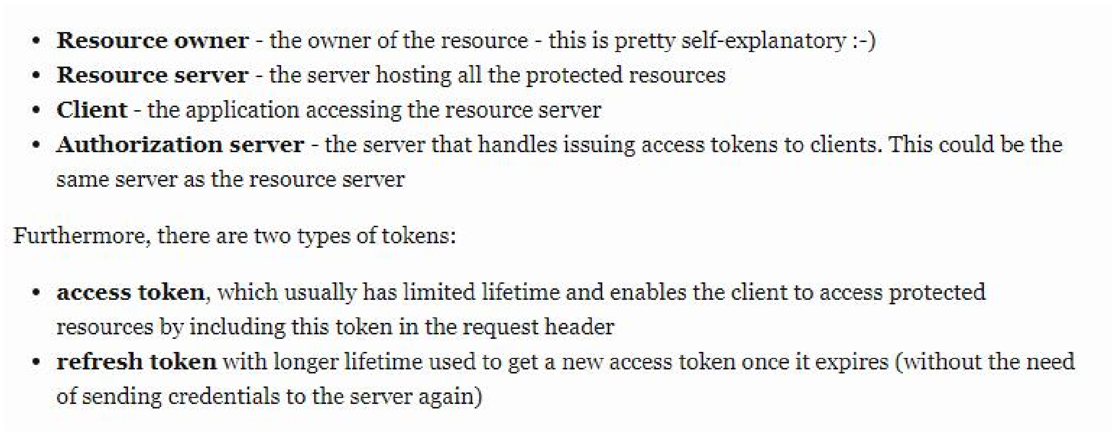
OAUTH-2:


OAUTH 2
1
App
4
Token
5
Token
ViVGreens
6
7
3
Token
2
User/Pass
ViVBook Auth Server
ViVBook
2 GB


OAUTH 2 Implicit
Username/Pass
Refresh Token 90 Days
Auth
Refresh Token
Access Token 10 Seconds
Your Service
Access Token



OAuth2 is a protocol (or authorization framework, as I prefer to refer to it) that describes a stateless authorization (that means we don't need to maintain sessions between clients and our server).



- **Resource owner** - the owner of the resource - this is pretty self-explanatory :-)
- **Resource server** - the server hosting all the protected resources
- **Client** - the application accessing the resource server
- **Authorization server** - the server that handles issuing access tokens to clients. This could be the same server as the resource server

Furthermore, there are two types of tokens:

- **access token**, which usually has limited lifetime and enables the client to access protected resources by including this token in the request header
- **refresh token** with longer lifetime used to get a new access token once it expires (without the need of sending credentials to the server again)

OAuth2 is a protocol (or authorization framework, as I prefer to refer to it) that describes a stateless authorization (that means we don't need to

maintain sessions between clients and our server).

http://websystique.com/spring-security/secure-spring-rest-api-using-oauth2/

Basic knowledge

Roles

OAuth2 defines 4 roles :

Resource Owner: generally yourself.

Resource Server: server hosting protected data (for example Google hosting your profile and personal information).

Client: application requesting access to a resource server (it can be your PHP website, a Javascript application or a mobile application).

Authorization Server: server issuing access token to the client. This token will be used for the client to request the resource server. This server can be the same as the authorization server (same physical server and same application), and it is often the case.

Tokens

Tokens are random strings generated by the authorization server and are issued when the client requests them.

There are 2 types of token:

Access Token: this is the most important because it allows the user data from being accessed by a third-party application. This token is sent by the client as a parameter or as a header in the request to the resource server. It has a limited lifetime, which is defined by the authorization server. It must be kept confidential as soon as possible but we will see that this is not always possible, especially when the client is a web browser that sends requests to the resource server via Javascript.

Refresh Token: this token is issued with the access token but unlike the latter, it is not sent in each request from the client to the resource server. It merely serves to be sent to the authorization server for renewing the access token when it has expired. For security reasons, it is not always possible to obtain this token. We will see later in what circumstances.

1. OAuth2 Roles

OAuth2 defines four roles:

resource owner:

Could be you. An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

resource server:

The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

client:

An application making protected resource requests on behalf of the resource owner and with its authorization. It could be a mobile app asking your permission to access your Facebook feeds, a REST client trying to access REST API, a web site [Stackoverflow e.g.] providing an alternative login option using Facebook account.

authorization server:

The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

2. OAuth2 Authorization Grant types

An authorization grant is a credential representing the resource owner's authorization (to access its protected resources) used by the client to obtain an access token. The specification defines four grant types:

authorization code

implicit

resource owner password credentials

client credentials

We will be using resource owner password credentials grant type. The reason is simple, we are not implementing a view which redirects us to a login page. Only the usage where a client [Postman or RestTemplate based Java client e.g.] have the Resource owner's credentials and they provide those credential [along with client credentials] to authorization server in order to eventually receive the access-token[and optionally refresh token], and then use that token to actually access the resources.

A common example is the GMail app [a client] on your smartphone which takes your credentials and use them to connect to GMail servers. It also shows that ‘Password Credentials Grant’ is best suited when both the client and the servers are from same company as the trust is there, you don’t want to provide your credentials to a third party.