Add instructor notes here.

Objective

- Spring Security
- Spring Security with Spring boot
- Outh2
- Outh2 with Spring Boot

Spring Security

- Spring Security is a Java/Java EE framework that provides authentication, authorization and other security features for enterprise applications.
- Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.
- Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements

Spring 4 Security Advantages

Spring 4 Security Framework provides the following Advantages:

- Open Source Security Framework
- Flexible, Easy to Develop and Unit Test the applications
- Declarative Security Programming
- Easy of Extendability
- Easy of Maintenance
- Takes full advantage of Spring DI(Dependency Injection) and AOP.
- We can develop Loosely-Coupled Applications.

Spring 4 Security Features

- Authentication and Authorization.
- Supports BASIC,Digest and Form-Based Authentication.
- Supports LDAP Authentication.
- Supports OpenID Authentication.
- Supports SSO (Single Sign-On) Implementation.
- Supports Cross-Site Request Forgery (CSRF) Implementation.
- Supports "Remember-Me" Feature through HTTP Cookies.
- Supports Implementation of ACLs
- Supports "Channel Security" that means automatically switching between HTTP and HTTPS.
- Supports I18N (Internationalisation).
- Supports JAAS (Java Authentication and Authorization Service).
- Supports Flow Authorization using Spring WebFlow Framework.
- Supports WS-Security using Spring Web Services.
- Supports Both XML Configuration and Annotations. Very Less or minimal XML Configuration.

Spring 4.x Security Framework supports the following New Features:

- Supports WebSocket Security.
- Supports Spring Data Integration.
- CSRF Token Argument Resolver.

Five Spring Security Concepts

1. Authentication
2. Authorization
3. Principle
4. Grant Authority
5. Roles

Authentication

Authentication is the process of proving an identity and it occurs when subjects provide appropriate credentials to prove their identity. For example, when a user provides the correct password with a username, the password proves that the user is the owner of the username.

Authorization

Once a user is identified and authenticated, they can be granted authorization based on their proven identity. It's important to point out that you can't have separate authorization without identification and authentication. In other words, if everyone logs on with the same account you can grant access to resources for everyone, or block access to resources for everyone. If everyone uses the same account, you can't differentiate between users. However, when users have been authenticated with different user accounts, they can be granted access to different resources based on their identity.

In summary, it's important to understand the differences between identification, authentication, and authorization when studying for security exams such as the Security+, SSCP, or CISSP exams. Identification occurs when a subject claims an identity (such as with a username) and authentication occurs when a subject proves their identity (such as with a password). Once the subject has a proven identity,

authorization techniques can grant or block access to objects based on their proven identities.

Spring Security Default

- Add mandatory authentication URL
- Add Login Form
- Handle Login error
- Create User & default password (Default username:user & password auto generated)
- User Name & password which we can provide

```
spring.security.user.home=Rahul
spring.security.user.password=123456
```

Spring Maven Dependency

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

- We need to add @EnableWebSecurity annotation to an @Configuration class to have the Spring Security configuration defined in any WebSecurityConfigurer or more likely by extending the WebSecurityConfigurerAdapter base class and overriding individual methods:

```
@Configuration
@EnableWebSecurity
```

Dependency

Spring Security –Maven Dependency

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends
WebSecurityConfigurerAdapter {
@Override
public void configure(WebSecurity web) throws
Exception {
// TODO Auto-generated method stub
super.configure(web);
} @Override
protected void configure(HttpSecurity http) throws
Exception {
// TODO Auto-generated method stub
super.configure(http);
}
```

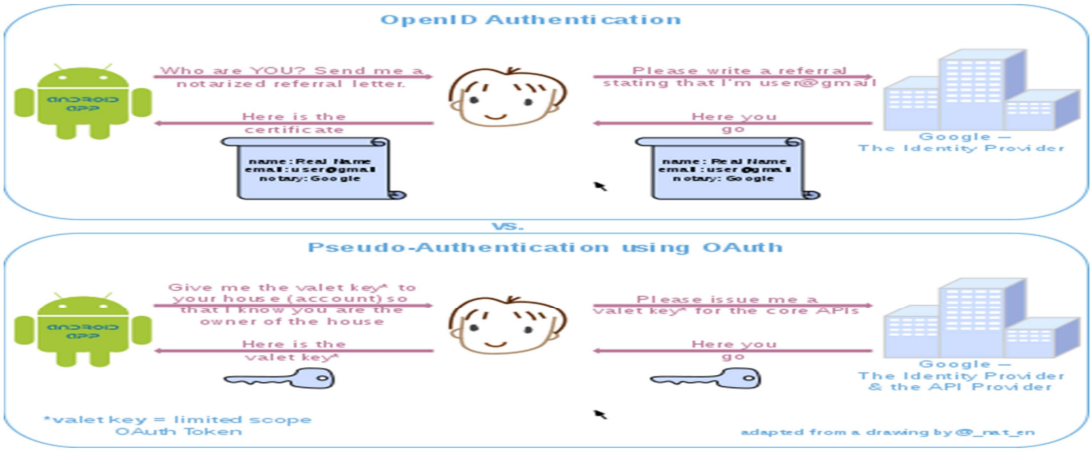
Demo

DemoSecurity

Outh 2.0

- OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.
- This mechanism is used by companies such as Amazon, Google, Facebook, Microsoft and Twitter to permit the users to share information about their accounts with third party applications or websites
- OAuth provides to clients a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP).

Outh 2.0



OAuth is an *authorization* protocol, rather than an *authentication* protocol. Using OAuth on its own as an authentication method may be referred to as pseudo-authentication.[*citation needed*] The following diagrams highlight the differences between using OpenID (specifically designed as an authentication protocol) and OAuth for authentication.

The communication flow in both processes is similar:

(Not pictured) The user requests a resource or site login from the application.

The site sees that the user is not authenticated. It formulates a request for the identity provider, encodes it, and sends it to the user as part of a redirect URL.

The user's browser requests the redirect URL for the identity provider, including the application's request

If necessary, the identity provider authenticates the user (perhaps by asking them for their username and password)

Once the identity provider is satisfied that the user is sufficiently authenticated, it processes the application's request, formulates a response, and sends that back to the user along with a redirect URL back to the application.

The user's browser requests the redirect URL that goes back to the application, including the identity provider's response

The application decodes the identity provider's response, and carries on accordingly.

(OAuth only) The response includes an access token which the application can use to gain direct access to the identity provider's services on the user's behalf.
The crucial difference is that in the OpenID *authentication* use case, the response from the identity provider is an assertion of identity; while in the OAuth *authorization* use case, the identity provider is also an API provider, and the response from the identity provider is an access token that may grant the application ongoing access to some of the identity provider's APIs, on the user's behalf. The access token acts as a kind of "valet key" that the application can include with its requests to the identity provider, which prove that it has permission from the user to access those APIs.
Because the identity provider typically (but not always) authenticates the user as part of the process of granting an OAuth access token, it's tempting to view a successful OAuth access token request as an authentication method itself.

Outh2.0

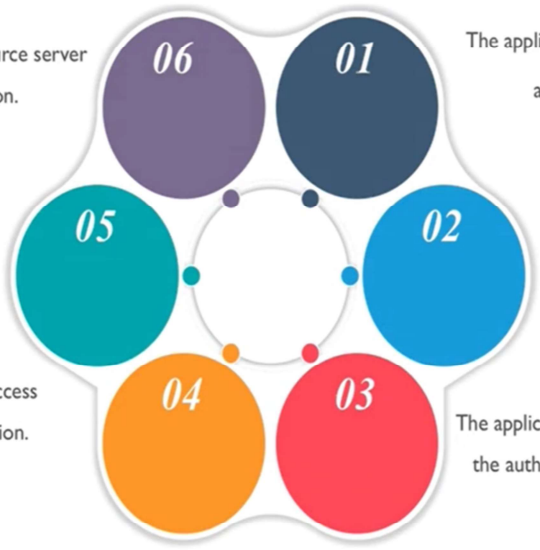
- OAuth2.0 is an **open authorization protocol**, which allows accessing the resources of the resource owner by enabling the client applications on HTTP services such as third provider party Facebook, GitHub, etc.
- It allows **sharing of resources** stored on one site with another site without using their credentials.

OAuth2 Client and Resource Server
There are four different *roles* within OAuth2 we need to consider:
Resource Owner — an entity that is able to grant access to its protected resources
Authorization Server — grants access tokens to *Clients* after successfully authenticating *Resource Owners* and obtaining their authorization
Resource Server — a component that requires an access token to allow, or at least consider, access to its resources
Client — an entity that is capable of obtaining access tokens from authorization servers
Annotating our configuration class with *@EnableResourceServer*, or *@EnableOAuth2Sso*, instructs Spring to configure components that transform our application into one of the latter two roles mentioned above.
The *@EnableResourceServer* annotation enables our application to behave as a *Resource Server* by configuring an *OAuth2AuthenticationProcessingFilter* and other equally important components.

Outh2.0 Flow


Client
(A)- Authorization Request
Resource Owner
(B)- Authorization Grant
(C)- Authorization Grant
Authorization Server
(D)- Access Token
(E)- Access Token
Resource Server
(F)- Protected Resource

How the Token works



06 — If the access token is valid, the resource server serves the resource to the application.

01 — The application requests authorization to access service resources.

05 — The application requests the resource from the resource server and presents the access token for authentication.

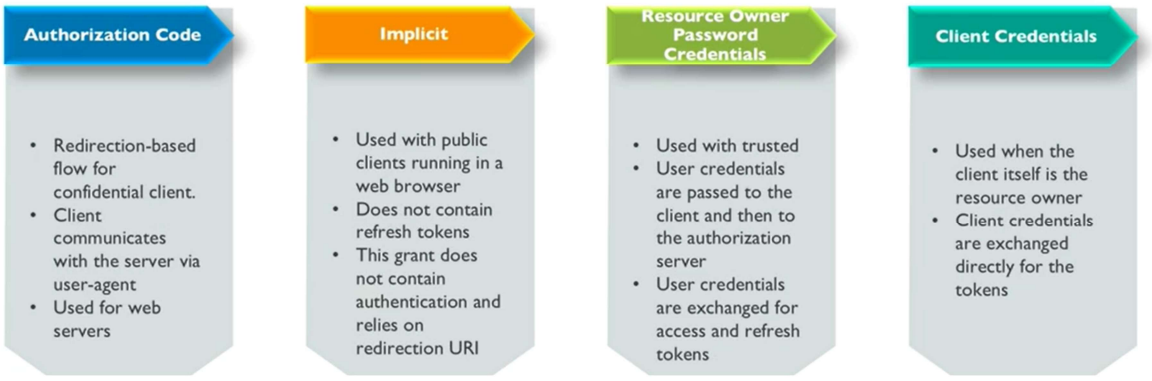
02 — The application receives an authorization grant if the request is authorized.

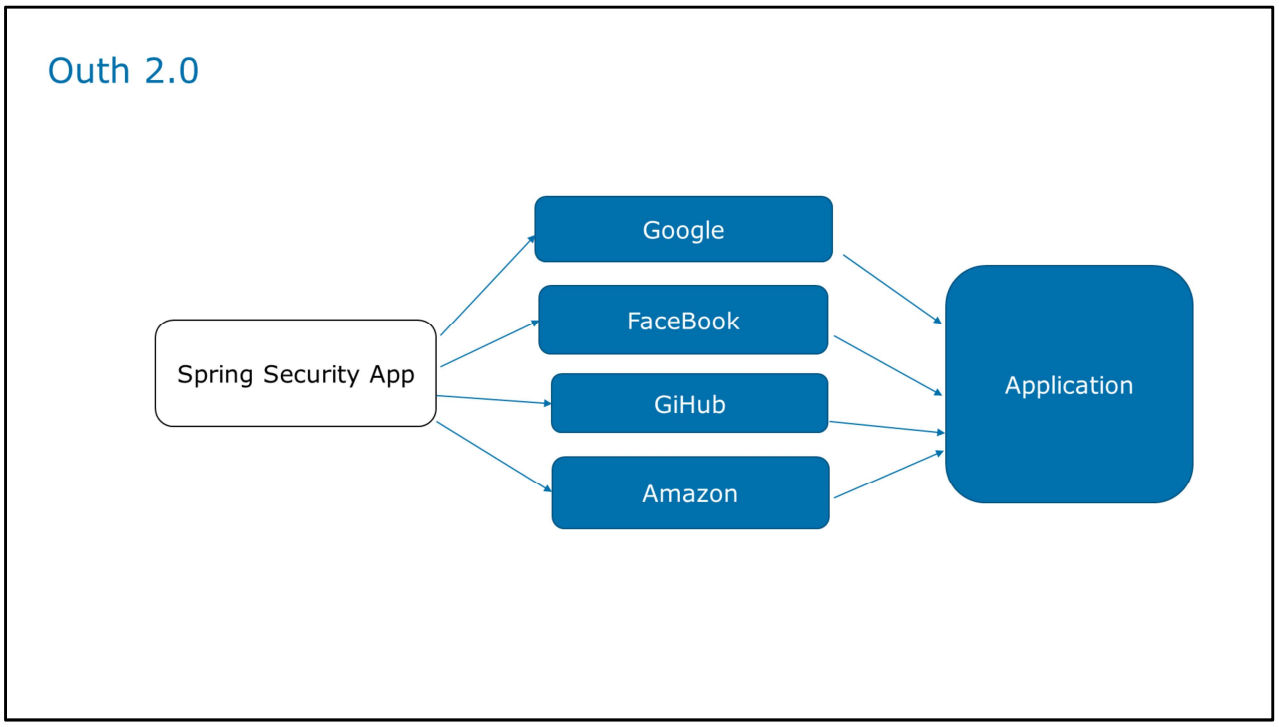
04 — The authorization server issues an access token to the application upon validation.

03 — The application requests an access token from the authorization server by presenting the authentication

Grant Type

- There are four different grant types defined by OAuth2. These grant types define the interactions between the client and the auth/resource server.





the *@EnableOAuth2Sso* annotation transforms our application into an OAuth2 client. It instructs Spring to configure an *OAuth2ClientAuthenticationProcessingFilter*, along with other components that our application needs to be capable of obtaining access tokens from an authorization server.
Take a look at the *SsoSecurityConfigurer* class for further details on what Spring configures for us.
Combining these annotations with some properties enables us to get things up and running quickly. Let's create two different applications to see them in action and how they can complement each other:
Our first application is going to be our edge node, a simple *Zuul* application that is going to use *@EnableOAuth2Sso* annotation. It's going to be responsible for authenticating users (with the help of an *Authorization Server*) and delegate incoming requests to other applications
The second application is going to use *@EnableResourceServer* annotation and will allow access to protected resources if the incoming requests contain a valid OAuth2 access token

Dependency

Maven Dependency

```
<groupId>org.springframework.security.oauth.boot</groupId>
<artifactId>spring-security-oauth2-autoconfigure</artifactId>
<version>2.1.2.RELEASE</version>
```

@EnableOAuth2Sso→The @EnableOAuth2Sso annotation enables OAuth2 Single Sign On (SSO). By default all the paths are secured. We can customize it using WebSecurityConfigurerAdapter in our Spring Security Java Configuration.

Outh 2.0

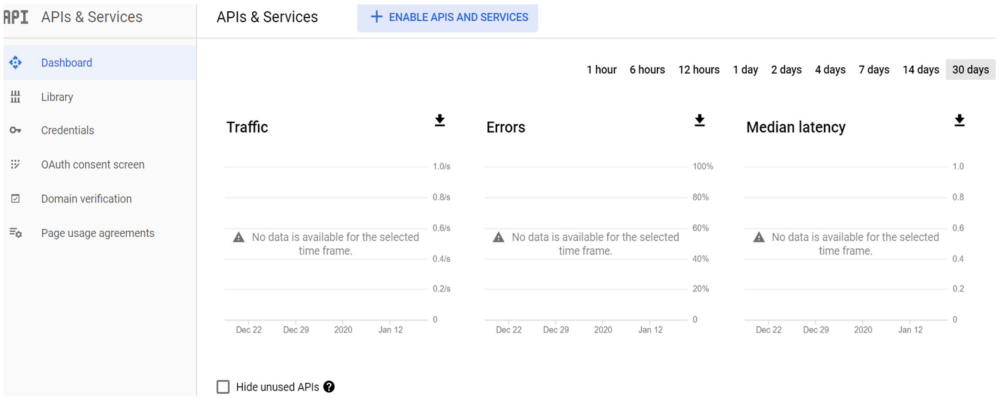
We can configure our Login & Logout pages, & which URL is going to be used by whom

```
@Configuration
@EnableOAuth2Sso
public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter {
        @Override
protected void configure(HttpSecurity http) throws
        Exception {
                http
            .authorizeRequests()
        .antMatchers("/", "/error**").permitAll()
            .anyRequest().authenticated()
        .and().logout().logoutUrl("/logout")
            .logoutSuccessUrl("/");

        }
    }
```

Outh 2.0 with Google

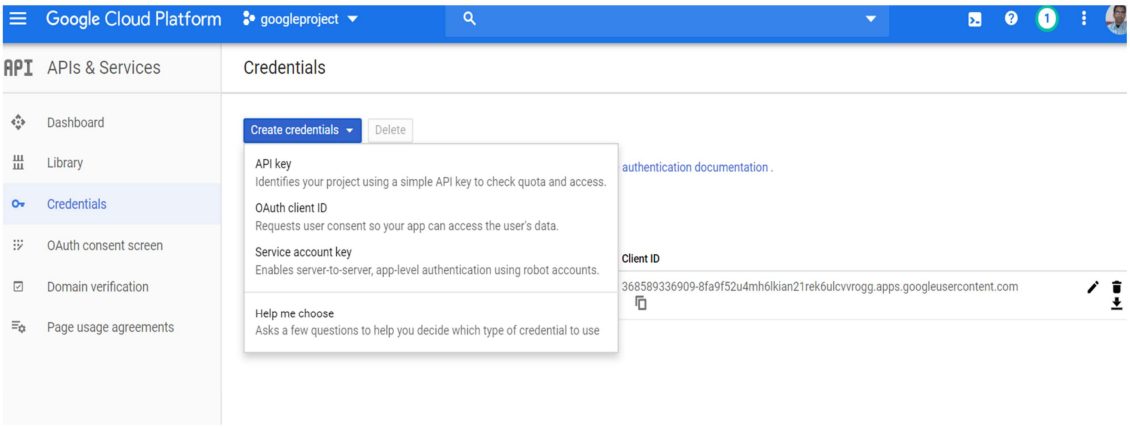
Login to https://console.cloud.google.com/



Outh 2.0 with Google

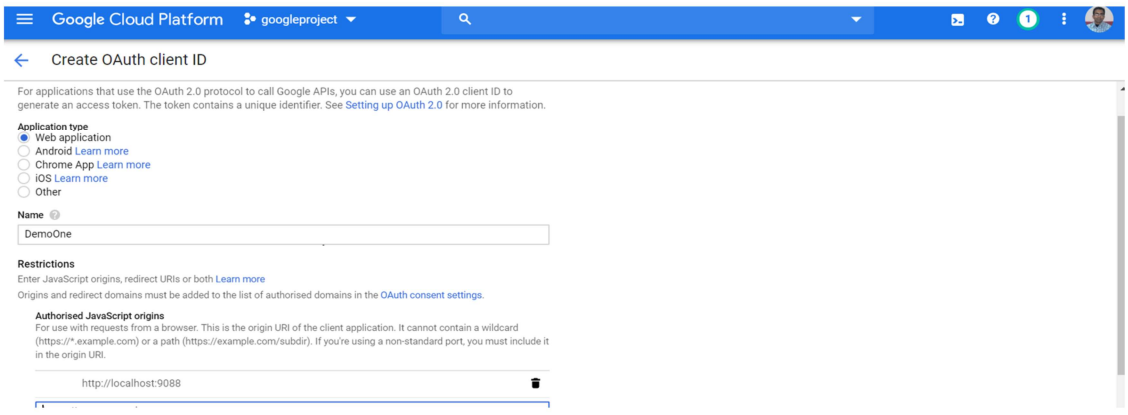
Go to credential

Click on Outh2.0 client Id

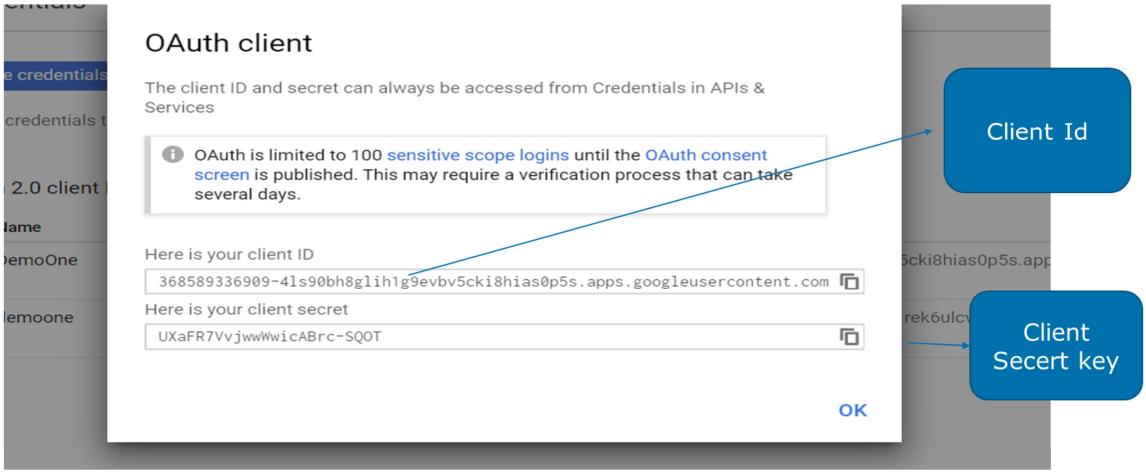


Outh 2.0 with Google

Click on Web Application , set your LocalHost port



Outh 2.0 with Google



Outh 2.0 with Google

```
security.oauth2.client.clientId=368589336909-
8fa9f52u4mh6lkian21rek6ulcvvrogg.apps.googleusercontent.com
security.oauth2.client.clientSecret=RrePb6ie2GXkNNv4C2V8HJGj
security.oauth2.client.accessTokenUri=https://www.googleapis.com/o
auth2/v3/token
security.oauth2.client.userAuthorizationUri=https://accounts.google.c
om/o/oauth2/auth
security.oauth2.client.tokenName=oauth_token
security.oauth2.client.authenticationScheme=query
security.oauth2.client.clientAuthenticationScheme=form
security.oauth2.client.scope=profile  email
security.oauth2.resource.userInfoUri=https://www.googleapis.com/us
erinfo/v2/me
security.oauth2.resource.preferTokenInfo=false
```

Client Id

Lab

Lab5