Prepared By Radha V Krishna

SPRINGBOOT SECURITY

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.

- Default Security SetUp


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

a default password is randomly generated and printed in the console log:

```
Using default security password: c8be15de-4488-4490-9dc6-fab3f91435c6
Username is user
```

In application.properties username and password can be set.

spring.security.user.name
spring.security.user.password

```
@SpringBootApplication(exclude = { SecurityAutoConfiguration.class })
public class SpringBootSecurityApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootSecurityApplication.class, args);
    }
}
```

Disables the auto configuration

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigu
re.security.SecurityAutoConfiguration
```

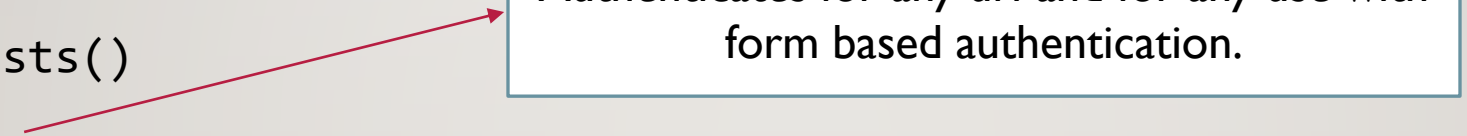
Userdefined credentials can be set in application.properties.

```
spring.security.user.name=user1
spring.security.user.password=password
```

```
@Configuration
@EnableWebSecurity
public class MySecurityConfiguration extends
WebSecurityConfigurerAdapter{

@Override
public void configure(HttpSecurity httpSecurity) throws
Exception
{
httpSecurity
.authorizeRequests()
.anyRequest()
.authenticated()
.and()
.formLogin();
}

}
```



```
@Override
public void configure(HttpSecurity
httpSecurity) throws Exception
{
httpSecurity
.authorizeRequests()
.antMatchers("/hello/user")

.hasRole("USER")
.antMatchers("/hello/admin")
.hasRole("ADMIN")
.anyRequest()
.authenticated()
.and()
.formLogin();

httpSecurity.csrf().disable();
}
```

```
@Override
public void
configure(AuthenticationManagerBuilder auth)
throws Exception
{
auth
.inMemoryAuthentication()
.withUser("user1")
.password("{noop}user1")
.roles("USER")
.and()
.withUser("admin")
.password("{noop}admin")
.roles("ADMIN");

}
```

```
//Example 2
public void configure(HttpSecurity
httpSecurity) throws Exception
{
httpSecurity
.authorizeRequests()
.antMatchers("/hello/user")

.hasAnyRole("USER","ADMIN")
.antMatchers("/hello/admin")
.hasRole("ADMIN")
.anyRequest()
.authenticated()
.and()
.formLogin();

httpSecurity.csrf().disable();
}
```

Authentication using Jdbc (h2 database)

Prepared By Radha V Krishna

```
@Autowired
private MyUserDetailsService userDetailsService;
@Override
public void configure(AuthenticationManagerBuilder auth) throws Exception
    {
        auth.userDetailsService(userDetailsService)
        /* calls loadUserByUsername(String username), returns the User object with other
values */
        .passwordEncoder(passwordEncoder());



    }

}
```

Enabling HTTPs

```yaml
server:
  port: 8443
  ssl:
    key-alias: springboot
    key-store: classpath:springboot.p12
    key-store-password: password
    key-password: password
```

keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype JKS -keystore springboot.jks -validity 3650 -storepass password

keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype PKCS12 -keystore springboot.p12 -validity 3650 -storepass password

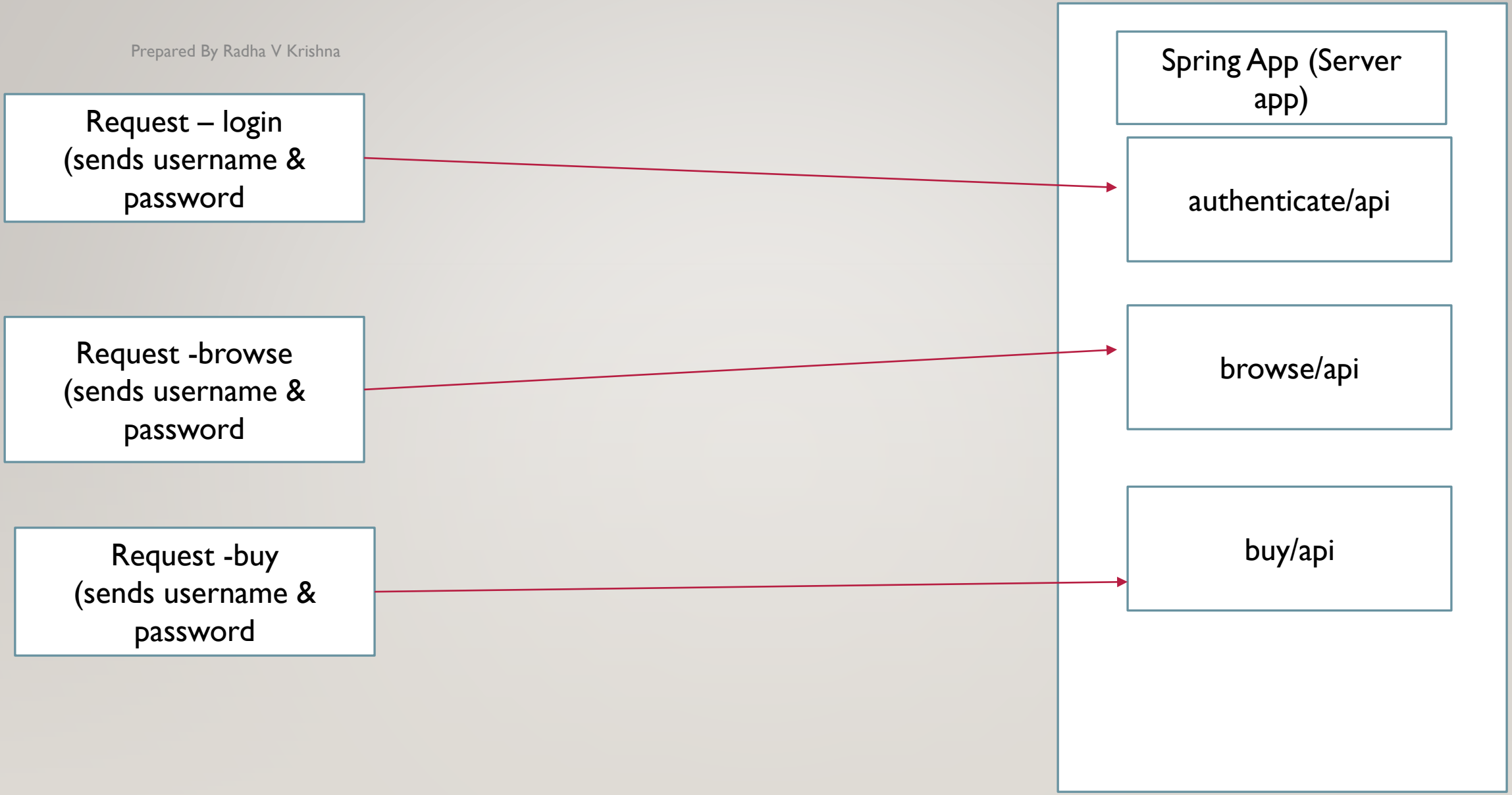
Public key cryptographic standards

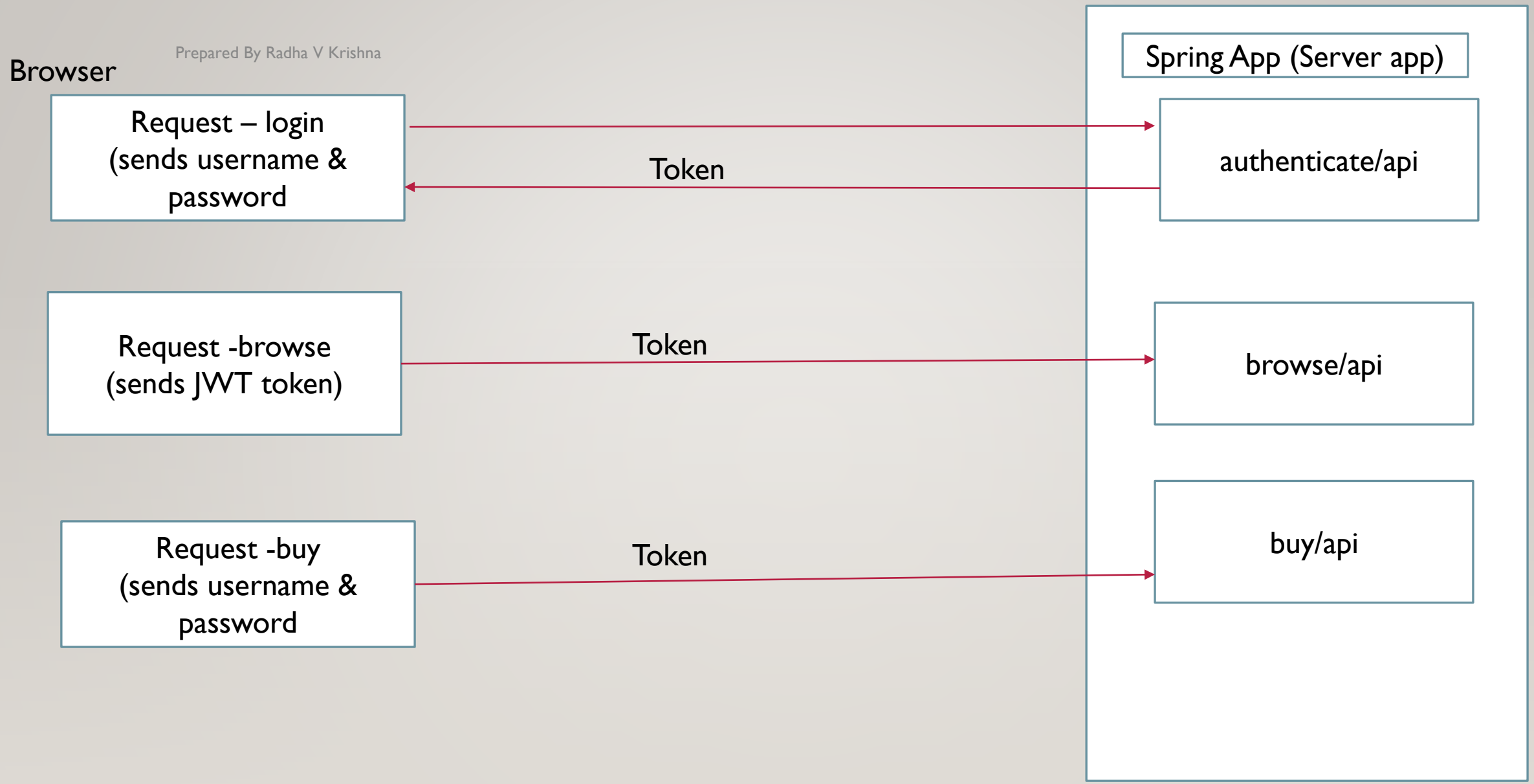
OAuth (Open Authorization) is an open standard for token-based authentication and authorization on the Internet. OAuth, which is pronounced "oh-auth," allows an end user's account information to be used by third-party services, such as Facebook, without exposing the user's password. OAuth acts as an intermediary on behalf of the end user, providing the service with an access token that authorizes specific account information to be shared. The process for obtaining the token is called a *flow.*

OAuth doesn’t share password data but instead uses authorization tokens to prove an identity between consumers and service providers. OAuth is an authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password.

JWT – JSON WEB TOKEN

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.


Request – login
(sends username & password
Request -browse
(sends username & password
Request -buy
(sends username & password
Spring App (Server app)
authenticate/api
browse/api
buy/api


Browser
Request – login
(sends username &
password
Token
Spring App (Server app)
authenticate/api
Request -browse
(sends JWT token)
Token
browse/api
Request -buy
(sends username &
password
Token
buy/api

Prepared By Radha V Krishna

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```


HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

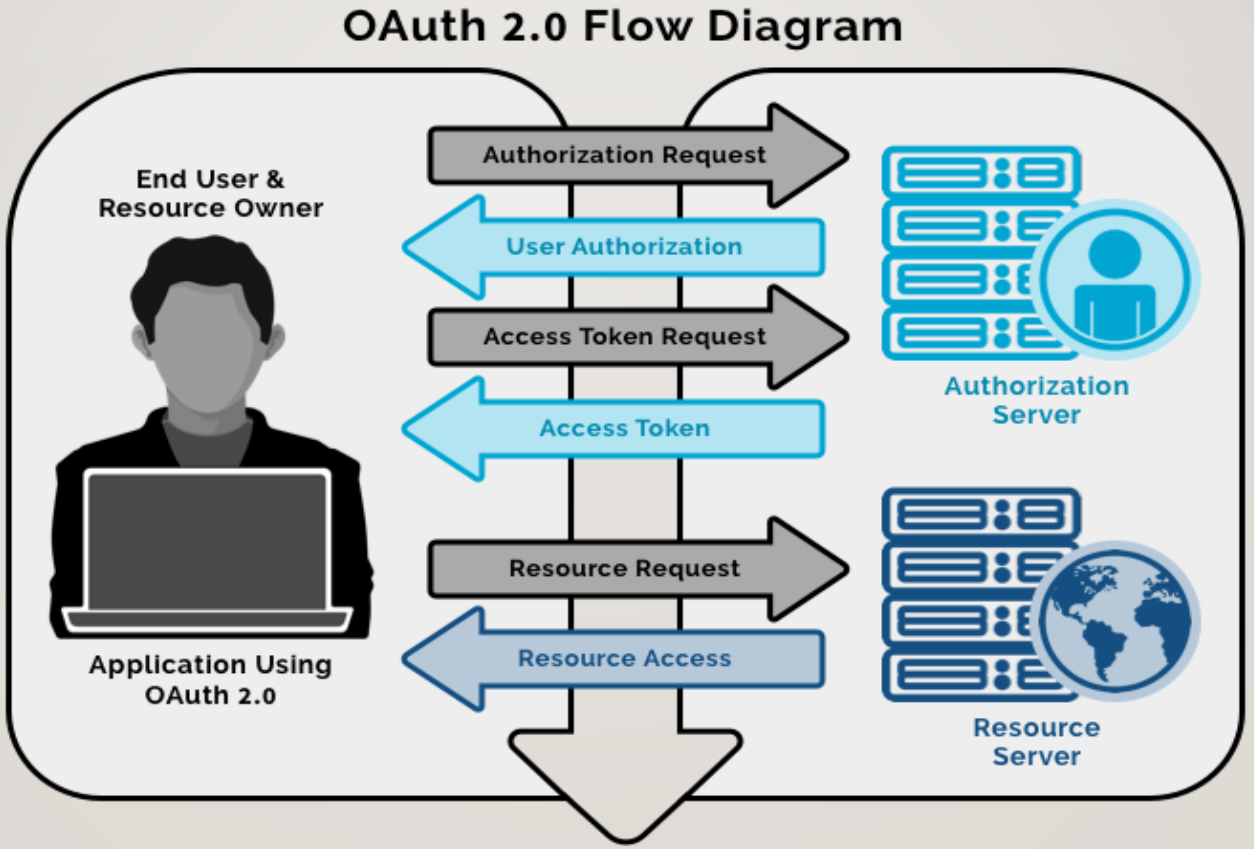

PAYLOAD: DATA


```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```


VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

The OAuth (open authorization) protocol was developed by the Internet Engineering Task Force and enables secure delegated access. It lets an application access a resource that is controlled by someone else (end user). This kind of access requires **Tokens,** which represent delegated right of access.



SSO – SINGLE SIGN ON

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-
oauth2</artifactId>
</dependency>
```

```
security:
  oauth2:
    client:
      clientId: fb1ff2ecaa7e72b4f75c
      clientSecret: c3d1cefad1616fdad55f60cf7811e0d4785eee3d
      accessTokenUri: https://github.com/login/oauth/access_token
      userAuthorizationUri:
https://github.com/login/oauth/authorize
      clientAuthenticationScheme: form
    resource:
      userInfoUri: https://api.github.com/user
      preferTokenInfo: false
```

```java
@RestController
public class MyController {

@GetMapping("/")
public String sayHelloOauth2(Principal principal)
{
return principal.getName()+" Welcome Oauth2 Cloud";
}
```

@EnableOAuth2Sso in the application