SPRING BOOT WITH SPRING SECURITY AND JWT

1. *Spring Boot OAuth2 Authorization server :*

The Spring Security OAuth stack offered the possibility of setting up an Authorization Server as a Spring Application. We then had to configure it to use JwtTokenStore so that we could use JWT tokens.

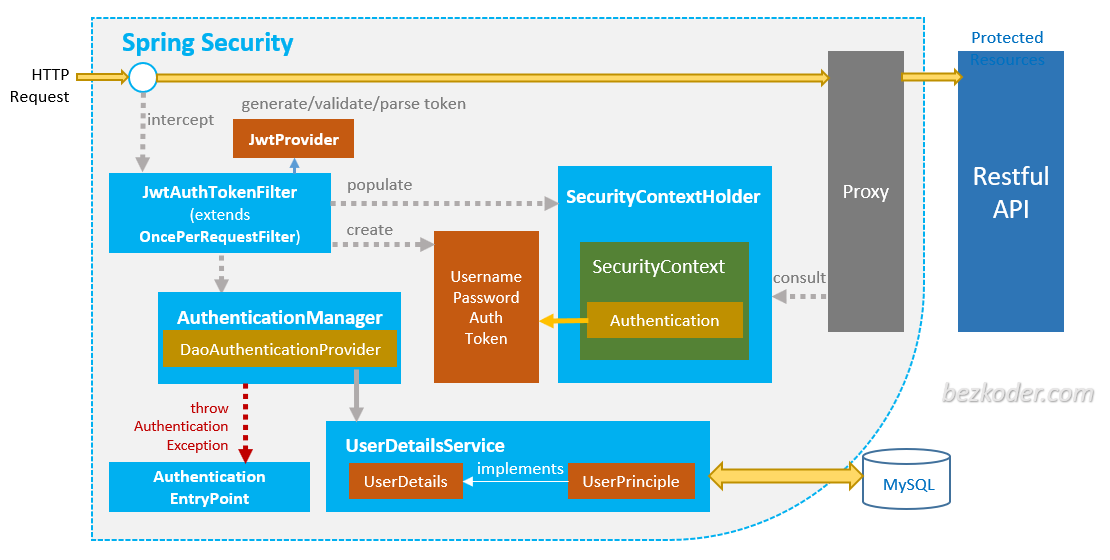
However, the OAuth stack has been deprecated by Spring and now we'll be using Keycloak as our Authorization Server.

**So this time, we'll set up our Authorization Server as**[an embedded Keycloak server in a Spring Boot app](https://www.baeldung.com/keycloak-embedded-in-a-spring-boot-application/). It issues JWT tokens by default, so there is no need for any other configuration in this regard.

These are the terms we need to address:

* **Authentication** refers to the process of verifying the identity of a user, based on provided credentials. A common example is entering a username and a password when you log in to a website.
* **Principle** refers to the currently authenticated user.
* **Granted authority** refers to the permission of the authenticated user.
* **Role** refers to a group of permissions of the authenticated user.

1. *Architecture :*



– SecurityContextHolder provides access to the SecurityContext.  
– SecurityContext holds the Authentication and possibly request-specific security information.  
– Authentication represents the principal which includes GrantedAuthority that reflects the application-wide permissions granted to a principal.  
– UserDetails contains necessary information to build an Authentication object from DAOs or other source of security data.  
– UserDetailsService helps to create a UserDetails from a String-based username and is usually used by AuthenticationProvider. UserDetailsService works with MySQL database via Spring Data JPA.  
– JwtAuthTokenFilter (extends OncePerRequestFilter) pre-processes HTTP request, from Token, create Authentication and populate it to SecurityContext.  
– JwtProvider validates, parses token String or generates token String from UserDetails.  
– UsernamePasswordAuthenticationToken gets username/password from login Request and combines into an instance of Authentication interface.  
– *AuthenticationManager* uses *DaoAuthenticationProvider* (with help of *UserDetailsService*& *PasswordEncoder)* to validate instance of*UsernamePasswordAuthenticationToken*, then returns a fully populated Authentication instance on successful authentication.  
– *SecurityContext* is established by calling SecurityContextHolder.getContext().setAuthentication(…​) with returned authentication object above.  
– *AuthenticationEntryPoint* handles *AuthenticationException.*  
– Access to Restful API is protected by *HTTPSecurity* and authorized with Method Security Expressions.

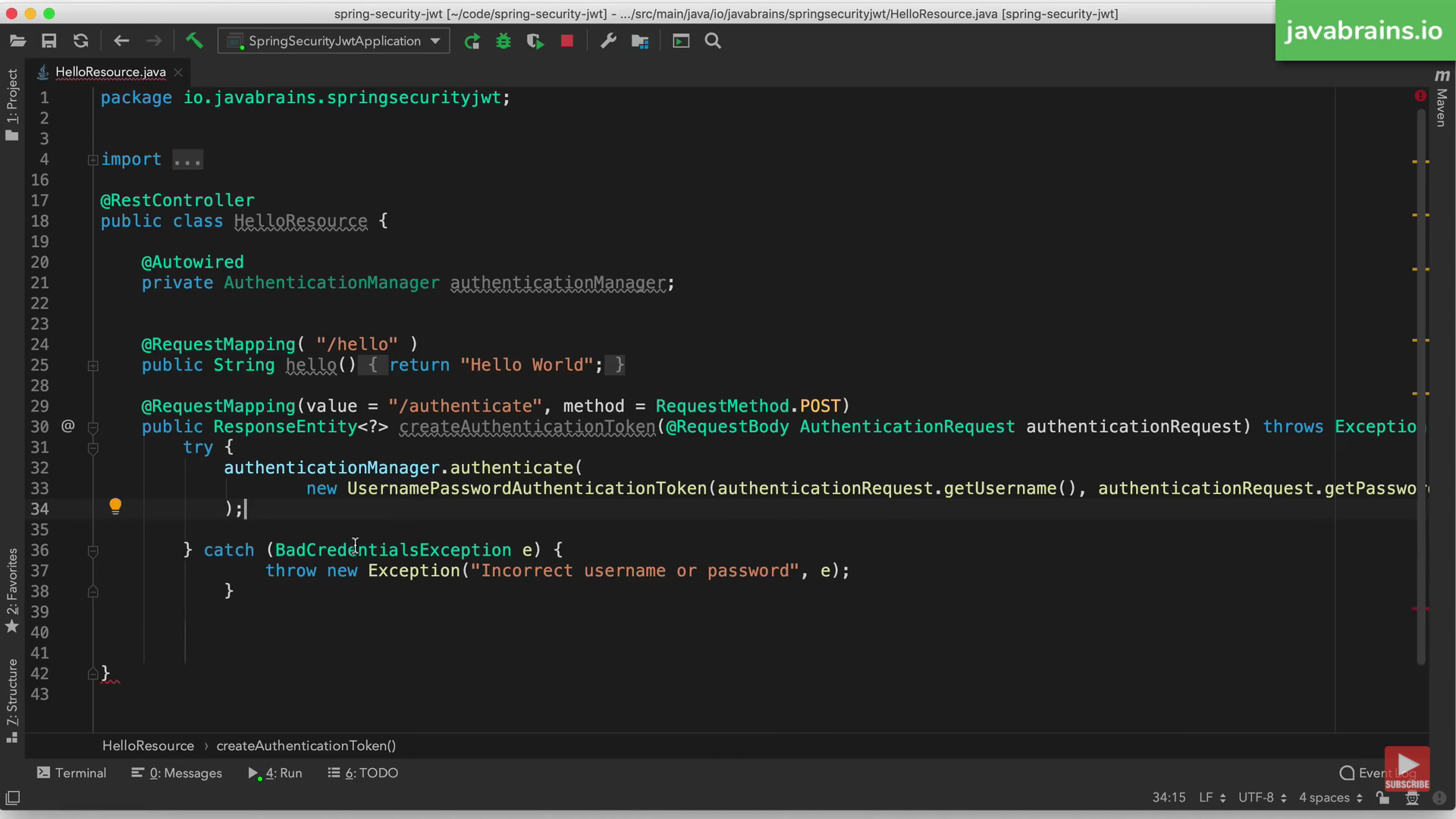
1. *Hands On :*



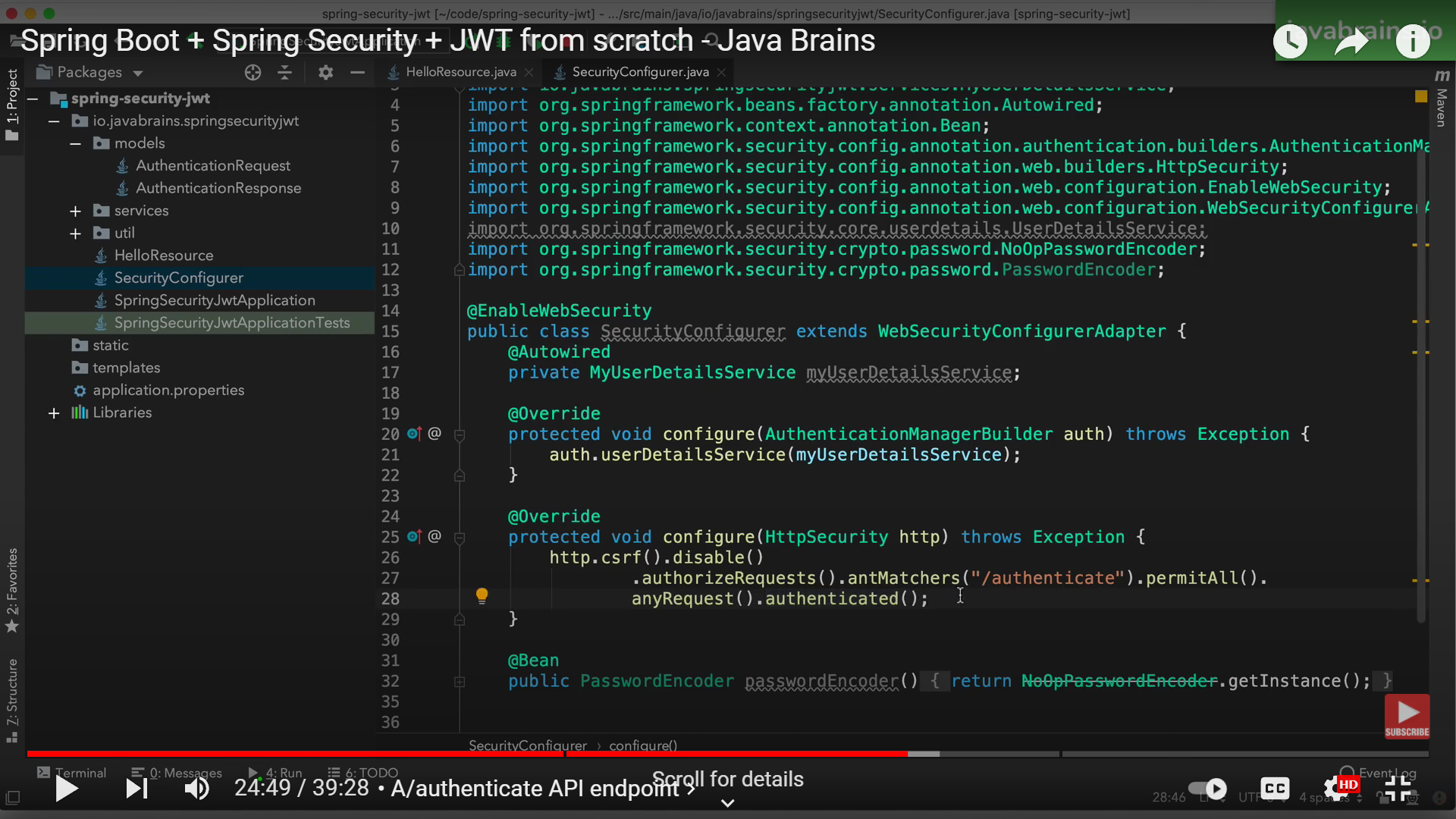
calling JWTs parser and security using SECRET\_KEY and set expiration time.



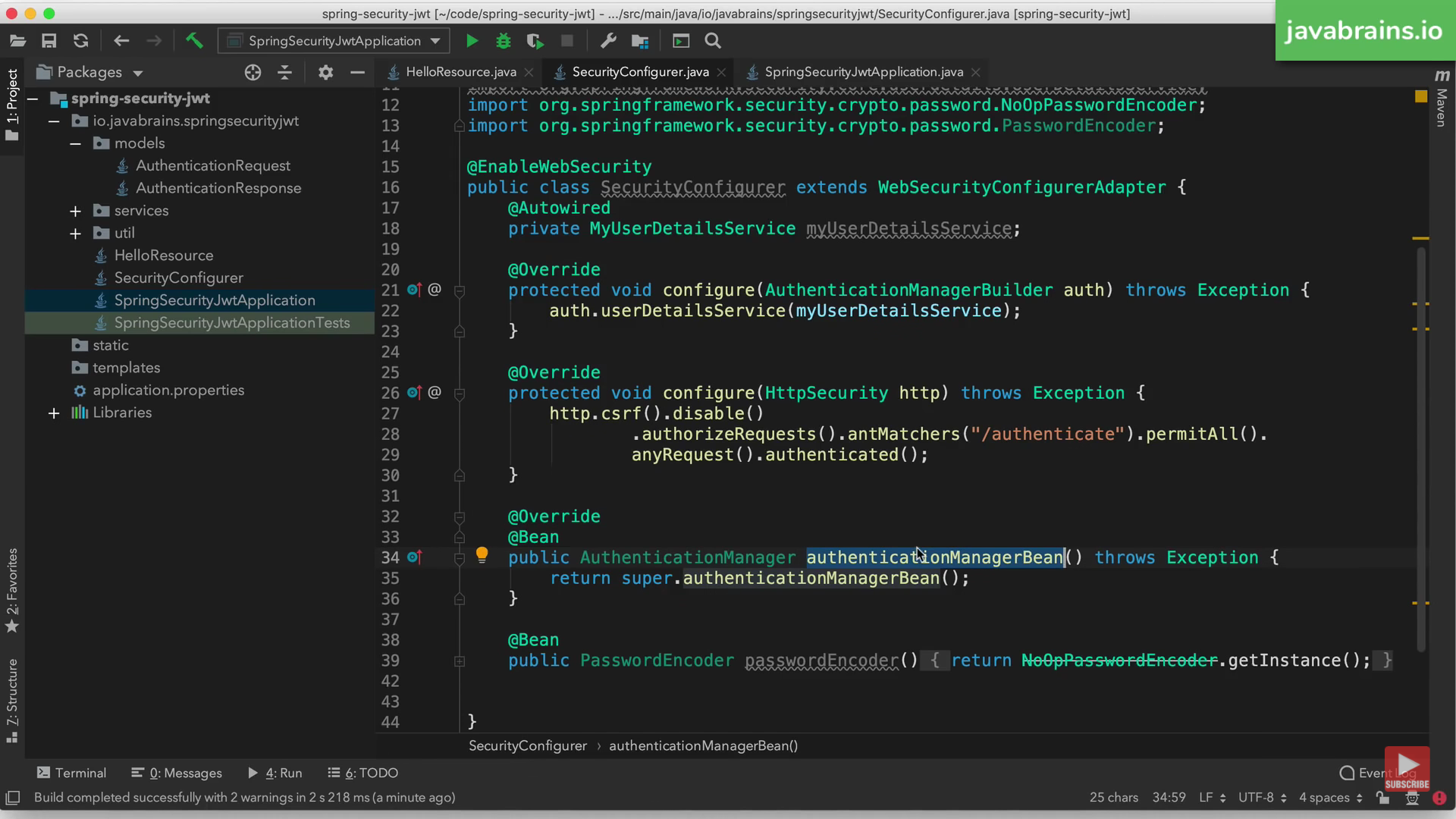
Authentication request class



This will return the actual jwt token

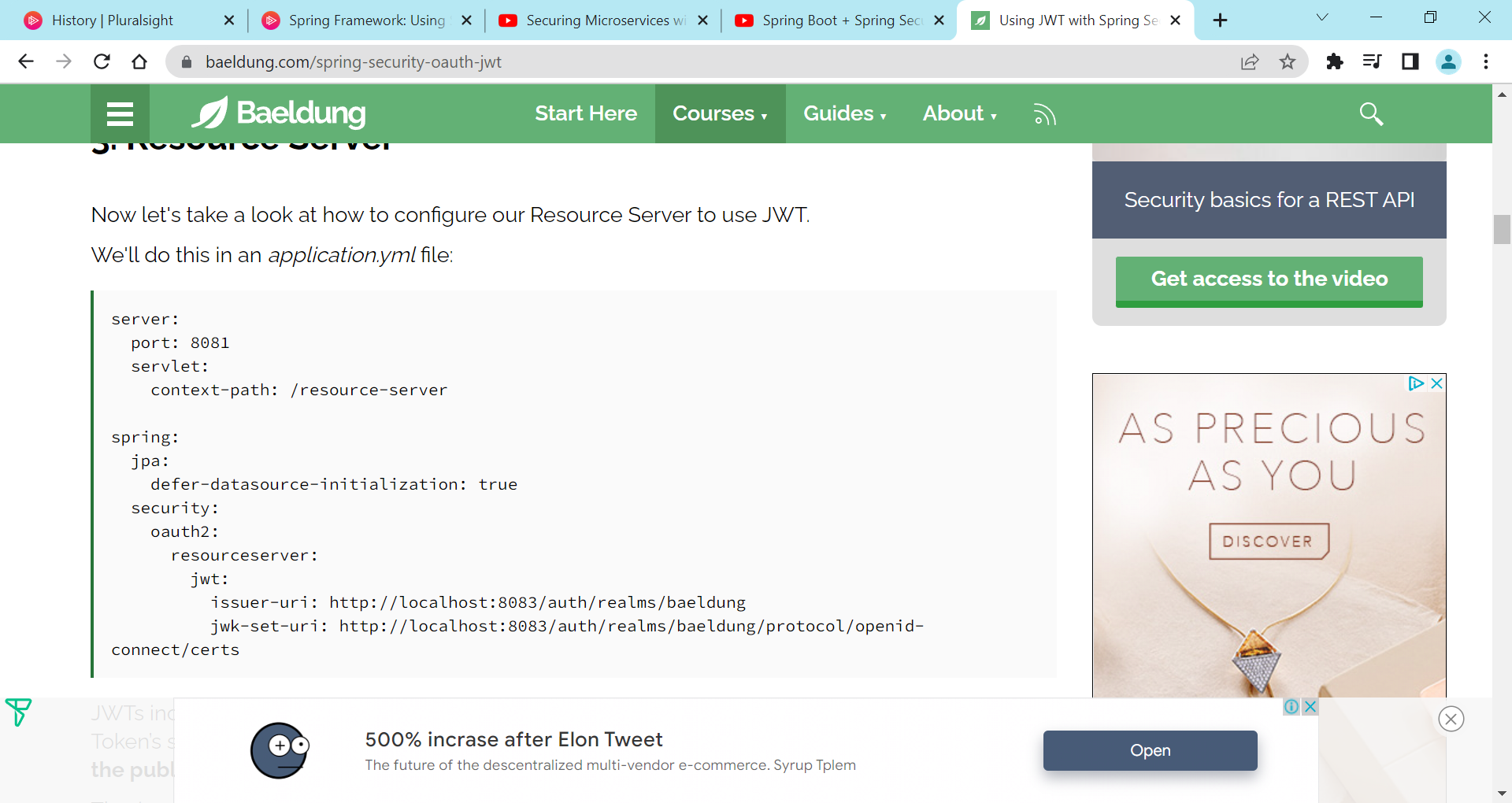


Security configuration method



Override authentication manager bean as it has been removed in updated version

Pointing Arrow



Application.yml file (port no. according to availablity)

