As a developer, build Authentication Provider in Spring Security.

**package** com.core.security.authentication.provider;

**import** com.javadevjournal.core.security.authentication.ExternalServiceAuthenticationToken;

**import** org.apache.commons.lang3.StringUtils;

**import** org.springframework.security.authentication.AuthenticationProvider;

**import** org.springframework.security.authentication.BadCredentialsException;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.AuthenticationException;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.core.userdetails.UserDetailsService;

**import** org.springframework.security.core.userdetails.UsernameNotFoundException;

**import** javax.annotation.Resource;

/\*\*

\* <p>A custom Authentication provider example. To create custom AuthenticationProvider, we need to implement the

\* AuthenticationProvider provide the implementation for the authenticate and support method.</p>

\*/

**public** **class** CustomAuthenticationProvider **implements** AuthenticationProvider {

@Resource

UserDetailsService userDetailsService;

/\*\*

\* <p> The authenticate method to authenticate the request. We will get the username from the Authentication object and will

\* use the custom **@userDetailsService** service to load the given user.</p>

\* **@param** authentication

\* **@return**

\* **@throws** AuthenticationException

\*/

@Override

**public** Authentication authenticate(Authentication authentication) **throws** AuthenticationException {

**final** String username = (authentication.getPrincipal() == **null**) ? "NONE\_PROVIDED" : authentication.getName();

**if** (StringUtils.isEmpty(username)) {

**throw** **new** BadCredentialsException("invalid login details");

}

// get user details using Spring security user details service

UserDetails user = **null**;

**try** {

user = userDetailsService.loadUserByUsername(username);

} **catch** (UsernameNotFoundException exception) {

**throw** **new** BadCredentialsException("invalid login details");

}

**return** createSuccessfulAuthentication(authentication, user);

}

**private** Authentication createSuccessfulAuthentication(**final** Authentication authentication, **final** UserDetails user) {

UsernamePasswordAuthenticationToken token = **new** UsernamePasswordAuthenticationToken(user.getUsername(), authentication.getCredentials(), user.getAuthorities());

token.setDetails(authentication.getDetails());

**return** token;

}

@Override

**public** **boolean** supports(Class << ? > authentication) {

**return** authentication.equals(ExternalServiceAuthenticationToken.**class**);

}

}

@EnableWebSecurity

**public** **class** AppSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests().anyRequest().authenticated();

}

/\*\*

\* Custom authentication provider. This authentication provider will authenticate the user with the help of

\* **@UserdetailsService.**

\* **@return**

\*/

@Bean

**public** CustomAuthenticationProvider authProvider() {

CustomAuthenticationProvider authenticationProvider = **new** CustomAuthenticationProvider();

**return** authenticationProvider;

}

/\*\*

\* Authentication manager which will be invoked by Spring security filter chain. This authentication

\* manager will delegate the work to the Authentication provider to

\* authenticate the user. Look out for the Custom authenitcation provider in the above section to see

\* how it works with this.

\* **@param** auth

\*/

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) {

auth.authenticationProvider(authProvider());

}

}