Spring Security

step 1)Add dependency in pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

step 2) create class E.g SecurityConfig extends WebSecurityConfigurerAdapter

It is mandatory to override its overloded protected void configure(AuthenticationManageBuilder auth) for (Authentication- validate user with username password )

And protected void configure(HttpSecurity http) for (Authorization--- to verifying what access do the have)

step 3) Annoted this security config class with @Configuration - mandatory

Setp 4) Its manadatory to use PasswordEncoder while authentication. Default password encoder given below.

PasswordEncoder encoder =

PasswordEncoderFactories.createDelegatingPasswordEncoder();

or

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

Annotation Description:-

@Configuration -@Configuration annotation indicates that a class declares one or more @Bean methods and

may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

@EnableWebSecurity annotation is crucial if we disable the default security configuration.

The application will fail to start if it's missing. So, the annotation is only optional if we're

just overriding the default behavior using a WebSecurityConfigurerAdapter.

Spring security core provide UserDetailsService and userDetails Interfaces for authentication

a)The UserDetailsService interface is used to retrieve user-related data.

It has one method named loadUserByUsername() which can be overridden to customize the process of finding the user.

It is used by the DaoAuthenticationProvider to load details about the user during authentication.

b) userDetails Provides core user information. Implementations are not used directly by Spring Security for security purposes.

Theysimply store user information which is later encapsulated into Authenticationobjects.

This allows non-security related user information (such as email addresses,telephone numbers etc) to be stored in a convenient location.

it contains 7 methods isCredentialsNonExpired,isEnabled,isAccountNonLocked,isAccountNonExpired etc.

Code:

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private static final Logger log= LogManager.getLogger(SecurityConfig.class);

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

PasswordEncoder encoder =

PasswordEncoderFactories.createDelegatingPasswordEncoder();

log.info("inside securityConfig configure");

auth.inMemoryAuthentication()

.withUser("user").password(encoder.encode("1234")).roles("USER")

.and()

.withUser("admin").password(encoder.encode("1234")).roles("USER", "ADMIN");

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

//HTTP Basic authentication

.httpBasic()

.and()

.authorizeRequests()

.antMatchers(HttpMethod.GET, "/books/\*\*").hasRole("USER")

.antMatchers(HttpMethod.POST, "/books").hasRole("ADMIN")

.antMatchers(HttpMethod.PUT, "/books/\*\*").hasRole("ADMIN")

.antMatchers(HttpMethod.PATCH, "/books/\*\*").hasRole("ADMIN")

.antMatchers(HttpMethod.DELETE, "/books/\*\*").hasRole("ADMIN")

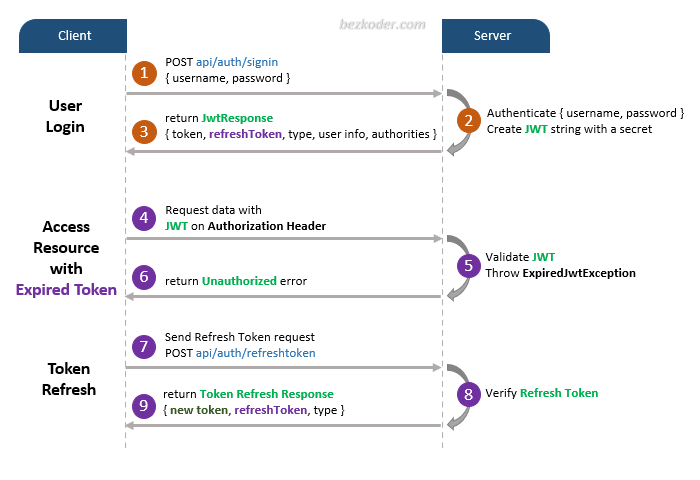
.and()

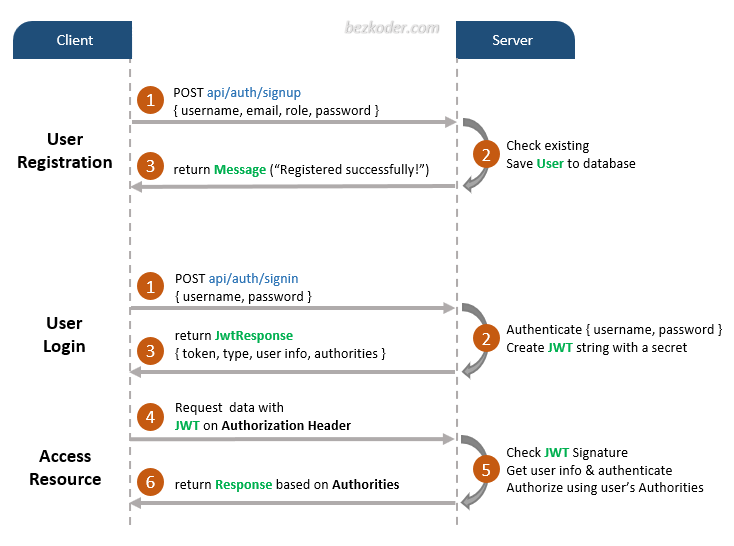
.csrf().disable()

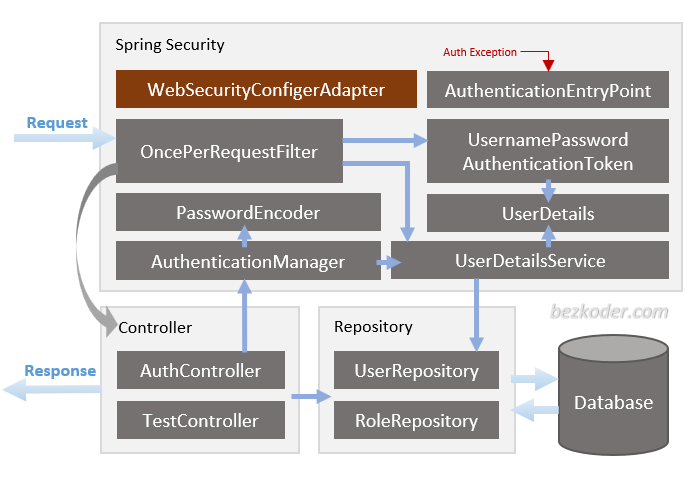
.formLogin().disable();

}

}







– [WebSecurityConfigurerAdapter](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/web/configuration/WebSecurityConfigurerAdapter.html) is the crux of our security implementation. It provides [HttpSecurity](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/" \l "jc-httpsecurity) configurations to configure cors, csrf, session management, rules for protected resources. We can also extend and customize the default configuration that contains the elements below.

– [UserDetailsService](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/" \l "tech-userdetailsservice) interface has a method to load User by username and returns a UserDetails object that Spring Security can use for authentication and validation.

– UserDetails contains necessary information (such as: username, password, authorities) to build an Authentication object.

– [UsernamePasswordAuthenticationToken](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/authentication/UsernamePasswordAuthenticationToken.html) gets {username, password} from login Request, AuthenticationManager will use it to authenticate a login account.

– [AuthenticationManager](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/" \l "core-services-authentication-manager) has a DaoAuthenticationProvider (with help of UserDetailsService & PasswordEncoder) to validate UsernamePasswordAuthenticationToken object. If successful, AuthenticationManager returns a fully populated Authentication object (including granted authorities).

– [OncePerRequestFilter](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/OncePerRequestFilter.html) makes a single execution for each request to our API. It provides a doFilterInternal() method that we will implement parsing & validating JWT, loading User details (using UserDetailsService), checking Authorizaion (using UsernamePasswordAuthenticationToken).

– [AuthenticationEntryPoint](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/AuthenticationEntryPoint.html) will catch authentication error.

****Repository**** contains UserRepository & RoleRepository to work with Database, will be imported into ****Controller****.

****Controller**** receives and handles request after it was filtered by OncePerRequestFilter.

– AuthController handles signup/login requests

– TestController has accessing protected resource methods with role based validations.

Understand the architecture deeply and grasp the overview more easier:  
[Spring Boot Architecture for JWT with Spring Security](https://bezkoder.com/spring-boot-jwt-mysql-spring-security-architecture/)

****security****: we configure Spring Security & implement Security Objects here.

* WebSecurityConfig extends WebSecurityConfigurerAdapter
* UserDetailsServiceImpl implements UserDetailsService
* UserDetailsImpl implements UserDetails
* AuthEntryPointJwt implements AuthenticationEntryPoint
* AuthTokenFilter extends OncePerRequestFilter
* JwtUtils provides methods for generating, parsing, validating JWT

****controllers**** handle signup/login requests & authorized requests.

* AuthController: @PostMapping(‘/signin’), @PostMapping(‘/signup’)
* TestController: @GetMapping(‘/api/test/all’), @GetMapping(‘/api/test/[role]’)

****repository**** has intefaces that extend Spring Data JPA JpaRepository to interact with Database.

* UserRepository extends JpaRepository<User, Long>
* RoleRepository extends JpaRepository<Role, Long>

****models**** defines two main models for Authentication (User) & Authorization (Role). They have many-to-many relationship.

* User: id, username, email, password, roles
* Role: id, name

****payload**** defines classes for Request and Response objects

We also have ****application.properties**** for configuring Spring Datasource, Spring Data JPA and App properties (such as JWT Secret string or Token expiration time).