Introduction:

[[TutsNode.com] - Spring Security Core Beginner to Guru\02 - Introduction to Spring Security\26127142-IntrotoSpringSecurity.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/02%20-%20Introduction%20to%20Spring%20Security/26127142-IntrotoSpringSecurity.pdf)

OWASP : **Open Web Application Security Project** (OWASP) is a nonprofit foundation dedicated to improving software security.

PDF Link: [[TutsNode.com] - Spring Security Core Beginner to Guru\02 - Introduction to Spring Security\25604716-CommonWebVulner.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/02%20-%20Introduction%20to%20Spring%20Security/25604716-CommonWebVulner.pdf) \*\*

Will be used in FE.

Cross Site Scripting (XSS) \*\*:

[[TutsNode.com] - Spring Security Core Beginner to Guru\02 - Introduction to Spring Security\25605506-CrossSiteScripting.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/02%20-%20Introduction%20to%20Spring%20Security/25605506-CrossSiteScripting.pdf)

Injectiong js scripts in input fields.

CSRF \*\*:

[[TutsNode.com] - Spring Security Core Beginner to Guru\02 - Introduction to Spring Security\25605760-CrossSiteForgery.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/02%20-%20Introduction%20to%20Spring%20Security/25605760-CrossSiteForgery.pdf)

Sending a redirect link which has cookies in it.

Csrf is a token that will be passed in the header or url to the server(from the Original Page). Bcz hacker is just redirecting, browser will not have the csrf token in url or header.

3.Http basic auth

User credentials are either be sending in **url** or **Header**

**url**: <https://username:password@www.xyz.com>

**Header**: Authorization: Basic <Base64 Encoded String> (encoded string = username:password (for testing, goto base64 encoder, then username:password and pass it to Authorization))

**Note:** Spring Security uses web mvc in the background.

3.6 testing spring security with Junit5

3.8 Spring security filter chain

[[TutsNode.com] - Spring Security Core Beginner to Guru\03 - HTTP Basic Auth\25683390-SpringSecurityFilterChain.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/03%20-%20HTTP%20Basic%20Auth/25683390-SpringSecurityFilterChain.pdf)

Spring Security filter chains will come before and after Dispatcher Servlet (front controller).

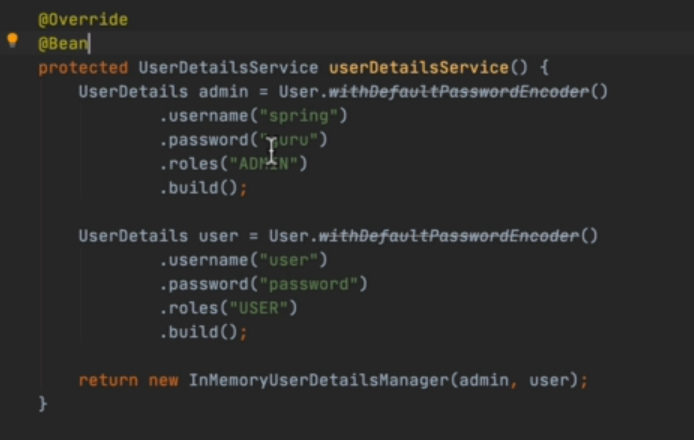
4.in memory authentication provider

4.1 Spring security Authentication flow:

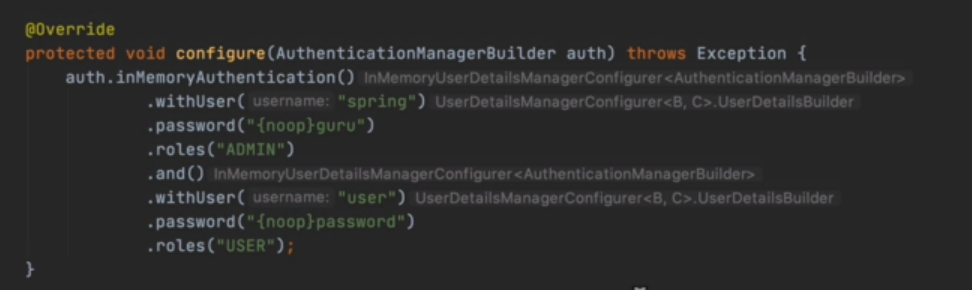
[[TutsNode.com] - Spring Security Core Beginner to Guru\05 - In Memory Authentication Provider\25732492-SpringSecAuthProcess.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/05%20-%20In%20Memory%20Authentication%20Provider/25732492-SpringSecAuthProcess.pdf)

We were doing configuration of basic inMemory http auth by, spring.security.user.name=”xx” spring.security.user.password=”xx”

We can do this manually by



Using Fluent Api technique:



Delegating Password Encoder

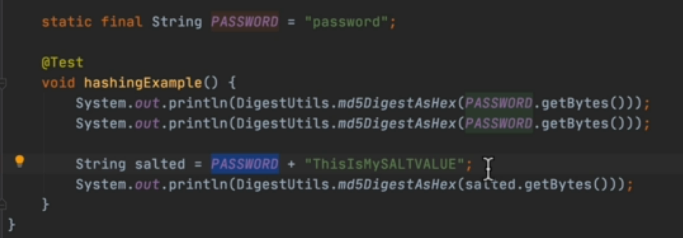
Allows storage of password hashes in multiple formats: {encodername}<somepasswordHashValue>.

Encodername= noop, bcrypt, …etc

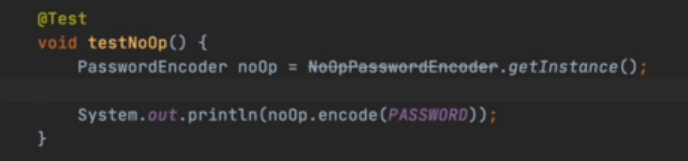
[[TutsNode.com] - Spring Security Core Beginner to Guru\06 - Password Security\25734930-PasswordEncoding.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/06%20-%20Password%20Security/25734930-PasswordEncoding.pdf) \*\*

4.3 Hashing Algorithms

MD5 hash & password salt: (not recommended)

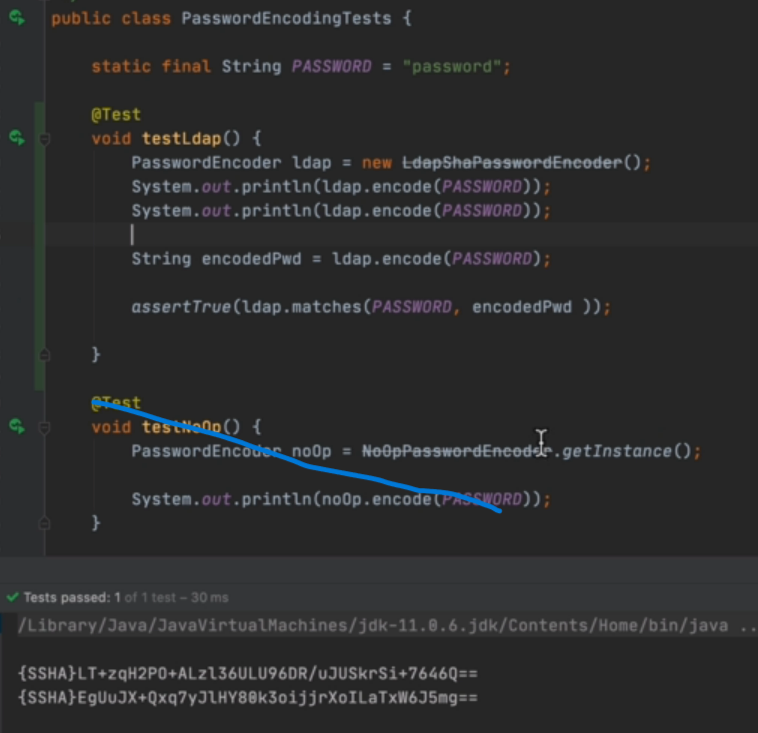


NoOp password Encoder: (not recommended)

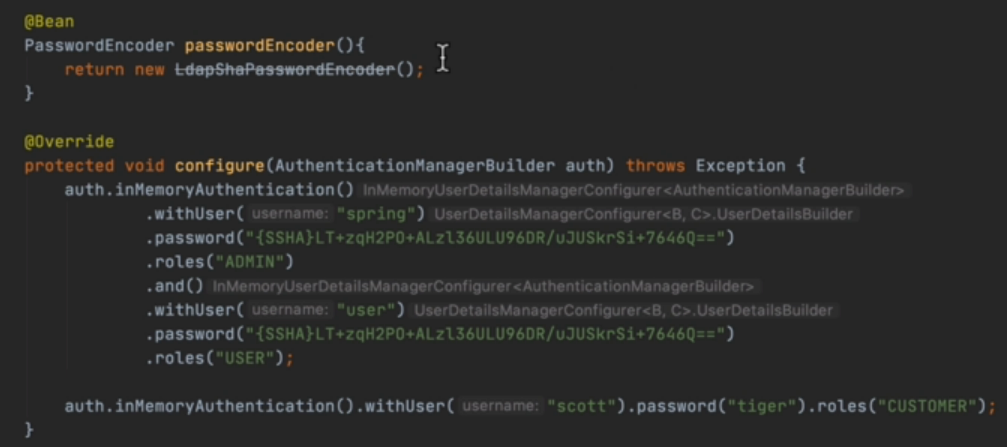


Ldap password Encoder: (not recommended)

-uses random salt



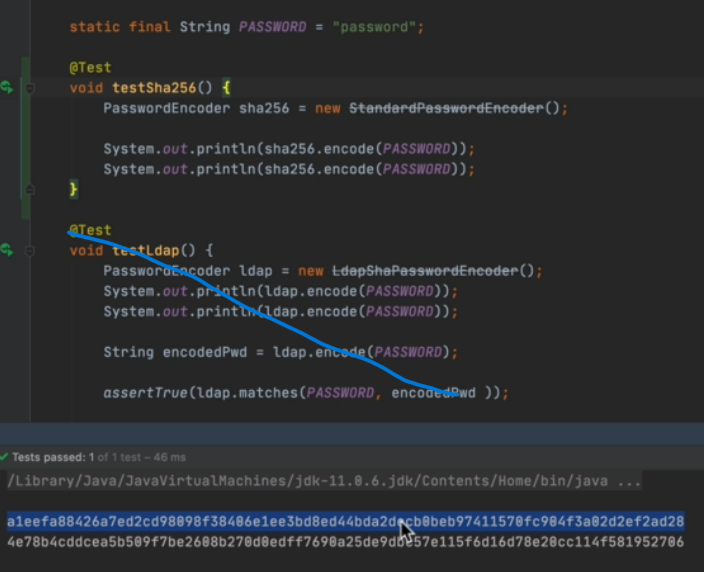
Declaring as Bean,



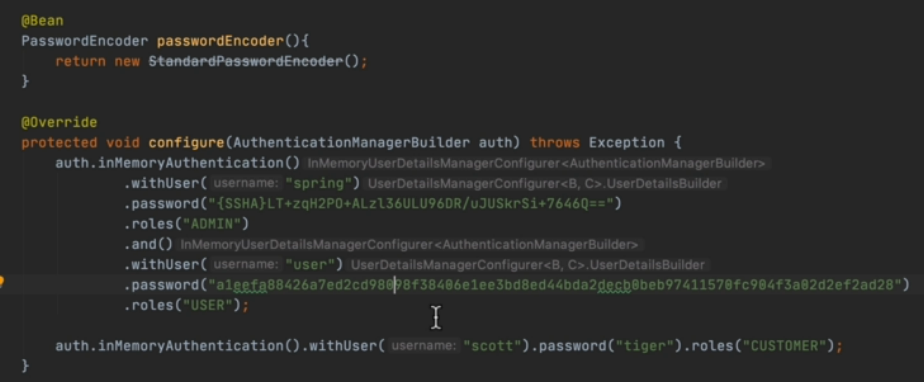
SHA-256 Password Encoder(not recommended)

-It was default in previous versions of spring security

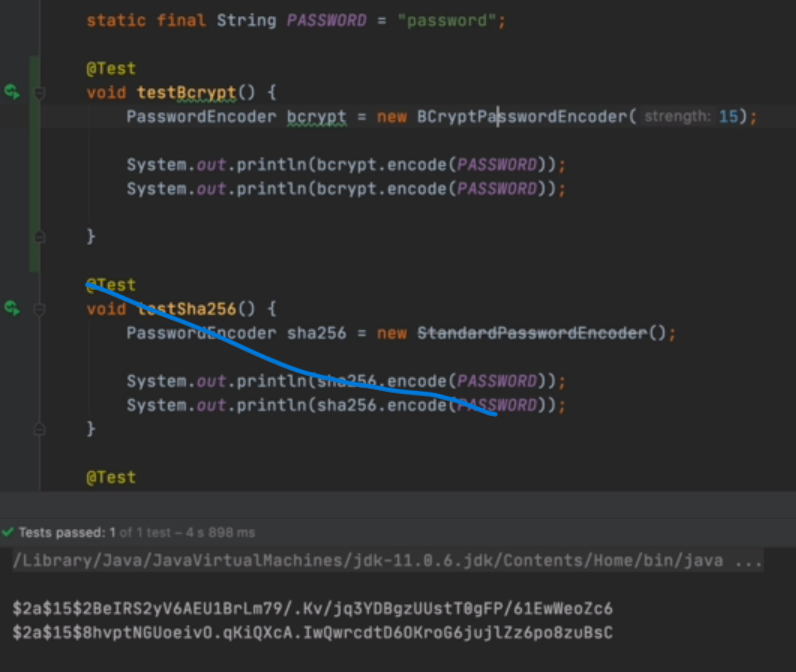
- Cons: tooFast in case of brut force attack. Brut Force attack requires more computational power. So other algorithms are slow compare to this.



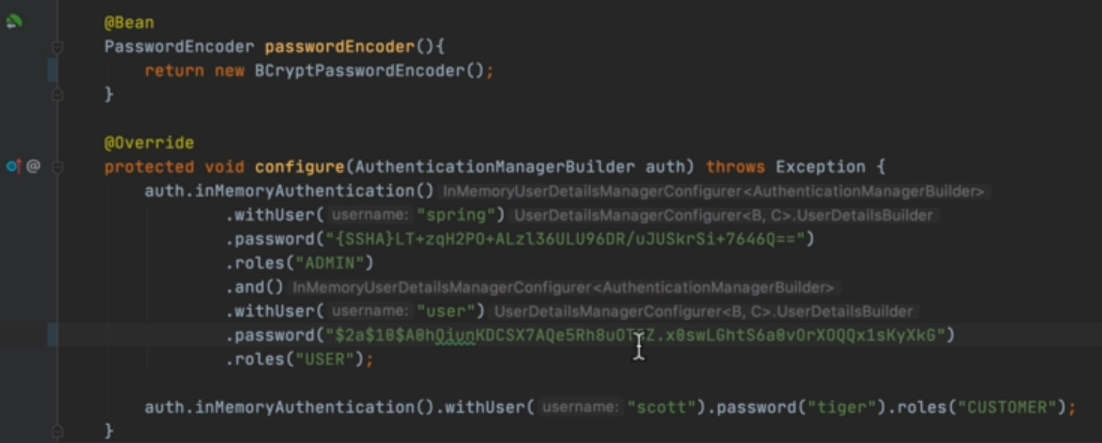
Declaring as Bean,



Bcrypt Password Encoder



Observation: It is decrypting the password that we passed in the config ( we are passing normal password in test cases, which is working)

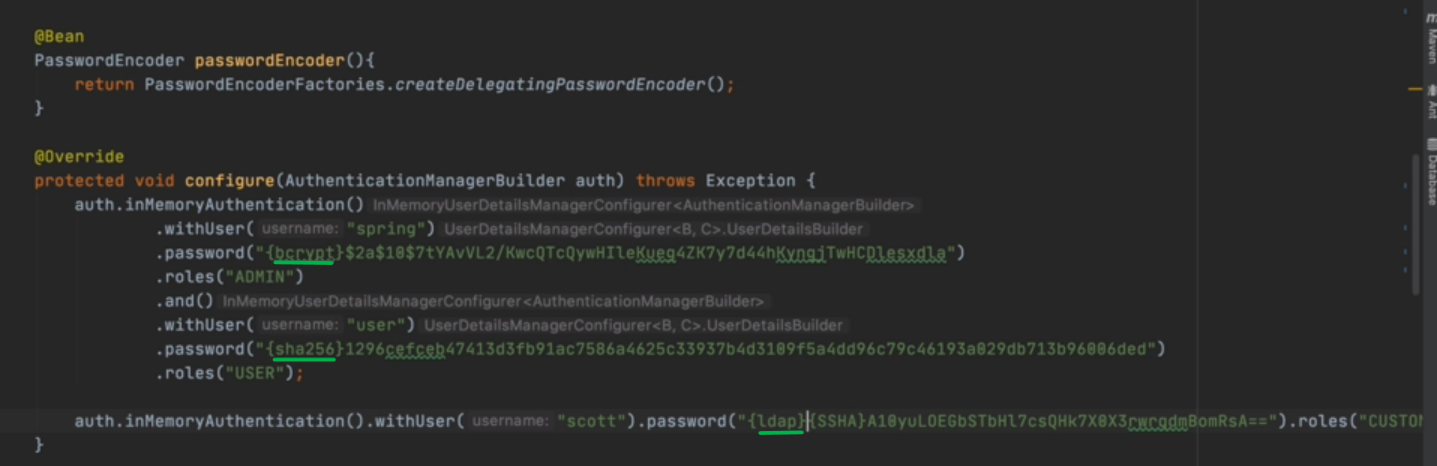


In testCase,



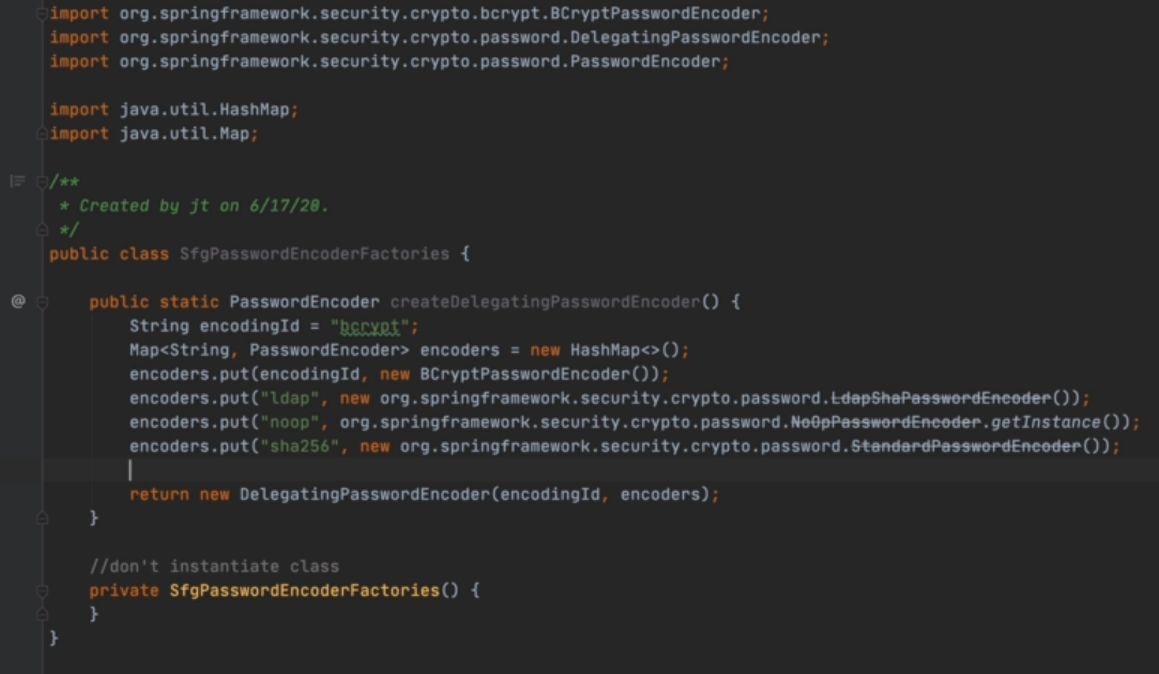
Delegating Password Encoder

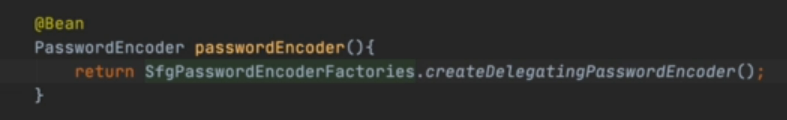
We can use multiple encoding algorithms at a time.



Custom Delegating Password Encoder

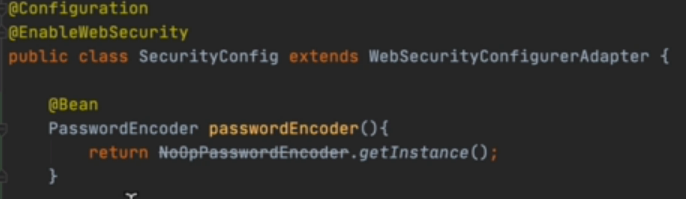
Create a class, here we are copying and paste the method “createDelegatingPasswordEncoder” from “PasswordEncoderFactoriesClass” and modify according to our need.





**Note:** By defining a specific password encoder bean, we can avoid doing {encodertype}”password” to “password” directly.

Example:



5. custom authentication filter

Intro: [pdfs\25757902-CustomAuthFilter.pdf](pdfs/25757902-CustomAuthFilter.pdf)

# Add a customAuthenticationfilter class extends AbstractAuthenticationProcessingFilter

**-attemptAuthentication** method is a abstract method inside AbstractAuthenticationProcessingFilter so we have to provide implementation for it (we have to implement logic for authentication).

-This class constructor takes api path for which it should apply this filter.

-The filter requires that you set the authenticationManager property. An AuthenticationManager is required to process the authentication request tokens created by implementing classes. (doing setAuthenticationManager in Img: filterConfig)

-This filter will intercept a request and attempt to perform authentication from that request if the request matches the [setRequiresAuthenticationRequestMatcher(RequestMatcher)](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/authentication/AbstractAuthenticationProcessingFilter.html#setRequiresAuthenticationRequestMatcher(org.springframework.security.web.util.matcher.RequestMatcher)).

-Authentication is performed by the [attemptAuthentication](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/authentication/AbstractAuthenticationProcessingFilter.html#attemptAuthentication(jakarta.servlet.http.HttpServletRequest,jakarta.servlet.http.HttpServletResponse)) method, which must be implemented by subclasses.

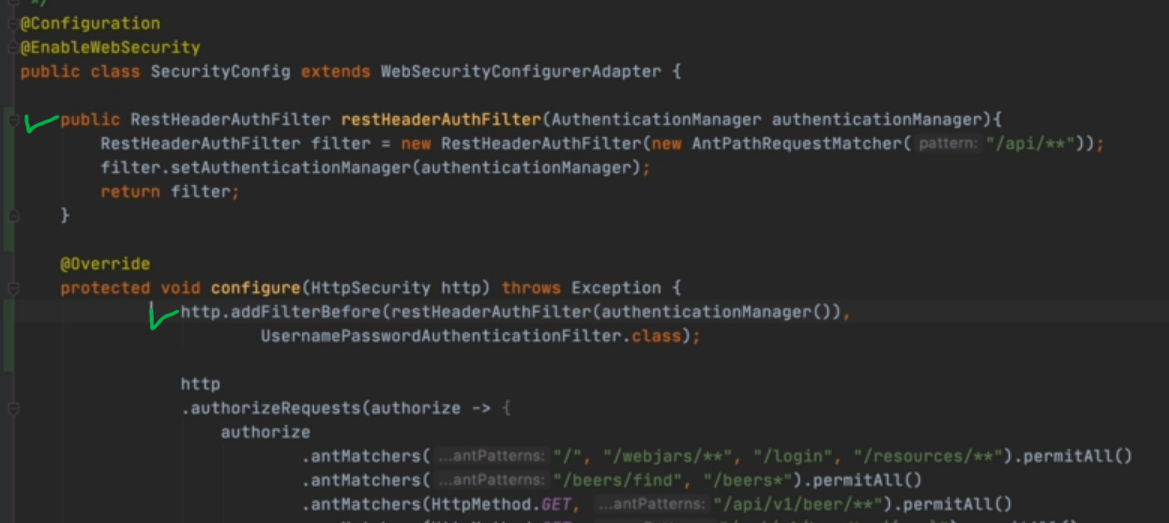
- If authentication is successful, the resulting [Authentication](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/Authentication.html) object will be placed into the SecurityContext for the current thread, which is guaranteed to have already been created by an earlier filter.

# Override dofilter(), successullAuthentication(), unsuccessfullAuthenticaton() to the same class

package guru.sfg.brewery.filters;  
  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.http.HttpStatus;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.AuthenticationException;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter;  
import org.springframework.security.web.util.matcher.RequestMatcher;  
import org.springframework.util.StringUtils;  
import javax.servlet.FilterChain;  
import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;   
  
@Slf4j  
public class RestHeaderAuthFilter extends AbstractAuthenticationProcessingFilter {  
  
 public RestHeaderAuthFilter(RequestMatcher requiresAuthenticationRequestMatcher) {  
 super(requiresAuthenticationRequestMatcher);  
 }  
  
 @Override  
 public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)  
 throws IOException, ServletException {  
  
 HttpServletRequest request = (HttpServletRequest) req;  
 HttpServletResponse response = (HttpServletResponse) res;  
  
// if (!requiresAuthentication(request, response)) {  
// chain.doFilter(request, response);  
// return;  
// }  
  
 if (logger.isDebugEnabled()) {  
 logger.debug("Request is to process authentication");  
 }  
  
 try{  
 //call attemptAuthenticate() method  
 Authentication authResult = attemptAuthentication(request, response);  
  
 if(authResult!=null) {  
 successfulAuthentication(request, response, chain, authResult);  
 chain.doFilter(request, response);  
 }  
 else {  
 //right now even if user is not authenticated we are not doing anything, just continuing the filter.  
 chain.doFilter(request, response);  
 }  
 }catch (AuthenticationException e){  
 unsuccessfulAuthentication(request, response, e);  
 }  
 }  
  
 @Override  
 public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException, IOException, ServletException {  
 String username = getUserName(request);  
 String password = getPassword(request);  
  
 if(username==null){  
 username="";  
 }  
 if(password == null){  
 password ="";  
 }  
  
 //Note: depending on the Authentication technique, it will compare the username and Password (http basic, custom userDetailService...)  
 //So if we configured Custom userDetailsService from db, it will take username and password in db.  
 UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(username, password);  
  
 if(!StringUtils.*isEmpty*(username)) {  
 return this.getAuthenticationManager().authenticate(token);  
 }  
 else {  
 return null;  
 }  
 }  
  
 @Override  
 protected void successfulAuthentication(HttpServletRequest request,  
 HttpServletResponse response, FilterChain chain, Authentication authResult)  
 throws IOException, ServletException {  
  
 if (logger.isDebugEnabled()) {  
 logger.debug("Authentication success. Updating SecurityContextHolder to contain: "  
 + authResult);  
 }  
  
 SecurityContextHolder.*getContext*().setAuthentication(authResult);  
 }  
  
 @Override  
 protected void unsuccessfulAuthentication(HttpServletRequest request,  
 HttpServletResponse response, AuthenticationException failed)  
 throws IOException, ServletException {  
 SecurityContextHolder.*clearContext*();  
  
 if (logger.isDebugEnabled()) {  
 logger.debug("Authentication request failed: " + failed.toString(), failed);  
 logger.debug("Updated SecurityContextHolder to contain null Authentication");  
 }  
  
 response.sendError(HttpStatus.*UNAUTHORIZED*.value(),  
 HttpStatus.*UNAUTHORIZED*.getReasonPhrase());  
 }  
  
 private String getPassword(HttpServletRequest request) {  
 return request.getHeader("username");  
 }  
  
 private String getUserName(HttpServletRequest request) {  
 return request.getHeader("password");  
 }  
}

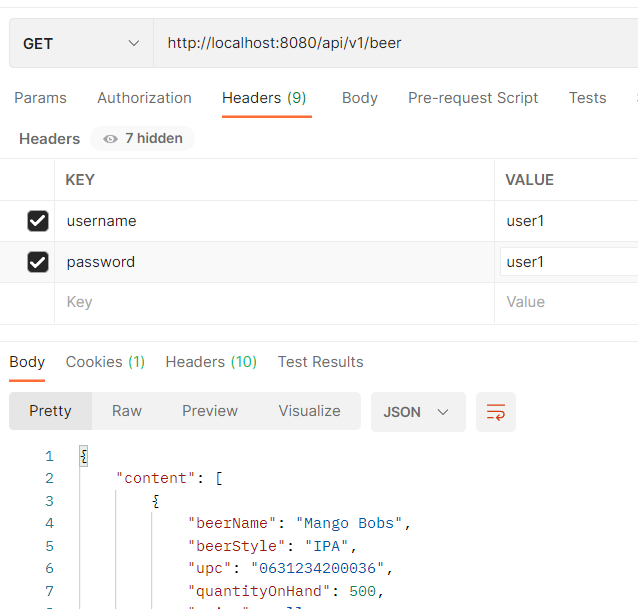
#Add this custom filter to security configuration

For all the matching api “/api/\*\*” this filter will apply.

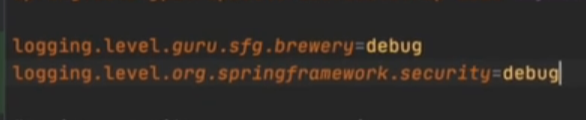


Img: filterConfig

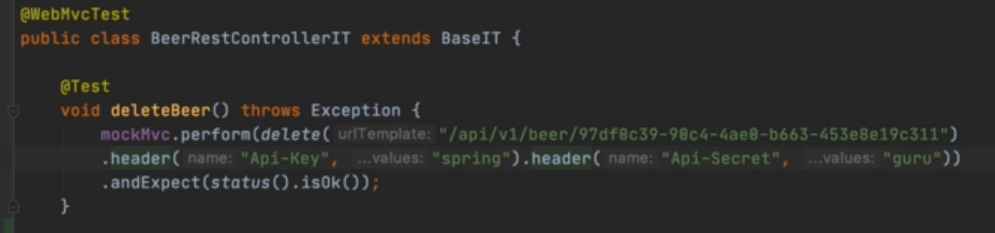
Observation: Hitting api in postman, we have added username and password in the header(in RestHeaderAuthFilter we are checking the headers for username and password). So based on the Authentication technique whether its httpbasic or custom userDetailsService …etc, it will fetch the username and password.



*Security debugging configuration:*



Checking in Test Case



6. database authentication

Intro: [pdfs\25761462-SpringSecurity.pdf](pdfs/25761462-SpringSecurity.pdf)

**Note:** User(c) implements UserDetails(i)

#Create User and Authority Class (similar to one in spring security)

User.java

package guru.sfg.brewery.domain.security;   
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Entity  
public class User {  
 @Id  
 @Column(name = "id", nullable = false)  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
  
 private String username;  
 private String password;  
  
 @Builder.Default //bcz @Builder will not pick up default properties  
 private boolean accountNonExpired = true;  
  
 @Builder.Default //bcz @Builder will not pick up default properties  
 private boolean accountNonLocked = true;  
  
 @Builder.Default //bcz @Builder will not pick up default properties  
 private boolean credentialsNonExpired = true;  
  
 @Builder.Default //bcz @Builder will not pick up default properties  
 private boolean enabled = true;  
  
 @Singular //provide a singular method for adding an authority (refer usage example below img:Singular)  
 @ManyToMany(cascade = CascadeType.*MERGE*)  
 @JoinTable(name="user\_authority",  
 joinColumns = {@JoinColumn(name = "USER\_ID", referencedColumnName = "ID")},  
 inverseJoinColumns = {@JoinColumn(name = "AUTHORITY\_ID", referencedColumnName = "ID")})  
 private Set<Authority> authorities;  
}

Authority.java

package guru.sfg.brewery.domain.security;   
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Entity  
public class Authority {  
  
 @Id  
 @Column(name = "id", nullable = false)  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String role;  
  
 @ManyToMany(mappedBy = "authorities")  
 private Set<User> users;  
}

usage of @Singular in entity class

Authority authorityAdmin = Authority.*builder*()  
 .role("ADMIN")  
 .build();  
  
Authority authorityUser = Authority.*builder*()  
 .role("USER")  
 .build();  
  
Authority authoritySuperAdmin = Authority.*builder*()  
 .role("SUPER\_ADMIN")  
 .build();  
  
User user1 = User.*builder*()  
 .username("user1")  
 .password(passwordEncoder.encode("user1"))  
 .authority(authorityAdmin) //here we are setting single entity instead of a Set<Authority>  
 .authority(authorityUser)  
 .build();

img: Singular

# Create UserRepo & AuthorityRepo

public interface UserRepository extends JpaRepository<User, Long> {  
 Optional<User> findByUsername(String username);  
}

public interface AuthorityRepository extends JpaRepository<Authority, Long> {  
}

bcz h2 db console does use frames, we need to set frame option, and we have to set it to same origin, so it function normally.

JFR :

(below classes (userDetails, User) are from java doc, jfr)

UserDetails:

public interface UserDetails extends Serializable {

Collection<? extends GrantedAuthority> getAuthorities();

String getPassword();

String getUsername();boolean isAccountNonExpired();boolean isAccountNonLocked();boolean isCredentialsNonExpired();  
boolean isEnabled();  
}

User:

public class User implements UserDetails, CredentialsContainer {  
  
 private static final long *serialVersionUID* = SpringSecurityCoreVersion.*SERIAL\_VERSION\_UID*;  
  
 private static final Log *logger* = LogFactory.*getLog*(User.class);  
  
 private String password;  
 private final String username;  
 private final Set<GrantedAuthority> authorities;  
 private final boolean accountNonExpired;  
 private final boolean accountNonLocked;  
 private final boolean credentialsNonExpired;  
 private final boolean enabled;  
public User(String username, String password,  
 Collection<? extends GrantedAuthority> authorities) {  
 this(username, password, true, true, true, true, authorities);  
 }

public User(String username, String password, boolean enabled,  
 boolean accountNonExpired, boolean credentialsNonExpired,  
 boolean accountNonLocked, Collection<? extends GrantedAuthority> authorities) {  
  
 if (((username == null) || "".equals(username)) || (password == null)) {  
 throw new IllegalArgumentException(  
 "Cannot pass null or empty values to constructor");  
 }  
  
 this.username = username;  
 this.password = password;  
 this.enabled = enabled;  
 this.accountNonExpired = accountNonExpired;  
 this.credentialsNonExpired = credentialsNonExpired;  
 this.accountNonLocked = accountNonLocked;  
 this.authorities = Collections.*unmodifiableSet*(*sortAuthorities*(authorities));  
 }  
  
 public Collection<GrantedAuthority> getAuthorities() {  
 return authorities;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public boolean isEnabled() {  
 return enabled;  
 }  
  
 public boolean isAccountNonExpired() {  
 return accountNonExpired;  
 }  
  
 public boolean isAccountNonLocked() {  
 return accountNonLocked;  
 }  
  
 public boolean isCredentialsNonExpired() {  
 return credentialsNonExpired;  
 }  
  
 public void eraseCredentials() {  
 password = null;  
 }

“””””””””

“””””””””

“”””””””” (refer doc)

6.7 Implementing UserDetailsService

@RequiredArgsConstructor  
@Service  
public class JpaUserDetailsService implements UserDetailsService {  
  
 private final UserRepository userRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username).orElseThrow(()-> new UsernameNotFoundException("username "+username+" not found"));  
 return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(),  
 user.getEnabled(), user.getAccountNonExpired(), user.getCredentialsNonExpired(),  
 user.getAccountNonLocked(), convertToGrantedAuthorites(user.getAuthorities()));  
 }  
  
 private Collection<? extends GrantedAuthority> convertToGrantedAuthorites(Set<Authority> authorities) {  
 if(authorities!=null && authorities.size()>0){  
 return authorities.stream()  
 .map(Authority::getRole)  
 .map(SimpleGrantedAuthority::new)  
 .collect(Collectors.*toList*());  
 }else{  
 return new HashSet<>();  
 }  
 }  
}