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).

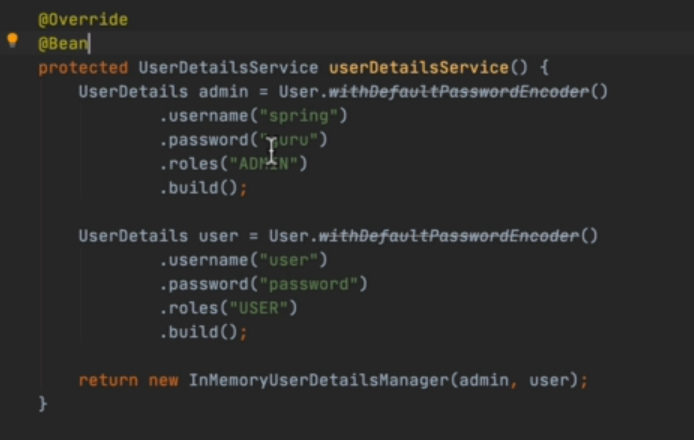
4.in memory authentication provider

4.1 Spring security Authentication flow:

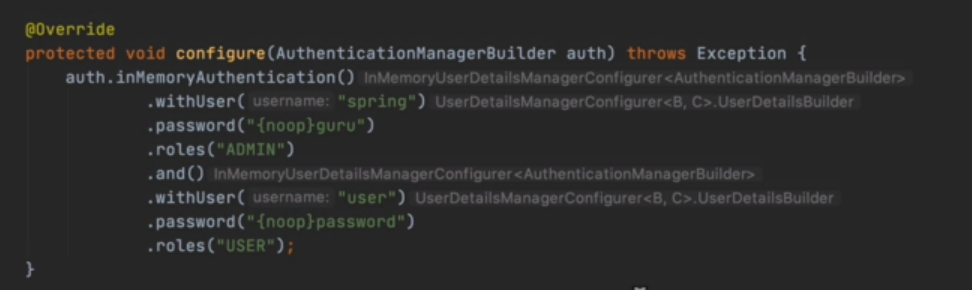
[[TutsNode.com] - Spring Security Core Beginner to Guru\05 - In Memory Authentication Provider\25732492-SpringSecAuthProcess.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/05%20-%20In%20Memory%20Authentication%20Provider/25732492-SpringSecAuthProcess.pdf)

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

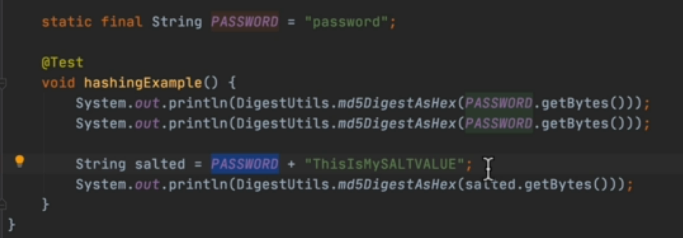
Allows storage of password hashes in multiple formats: {encodername}<somepasswordHashValue>.

Encodername= noop, bcrypt, …etc

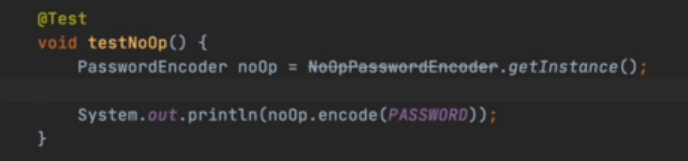
[[TutsNode.com] - Spring Security Core Beginner to Guru\06 - Password Security\25734930-PasswordEncoding.pdf](%5bTutsNode.com%5d%20-%20Spring%20Security%20Core%20Beginner%20to%20Guru/06%20-%20Password%20Security/25734930-PasswordEncoding.pdf) \*\*

4.3 Hashing Algorithms

MD5 hash & password salt: (not recommended)

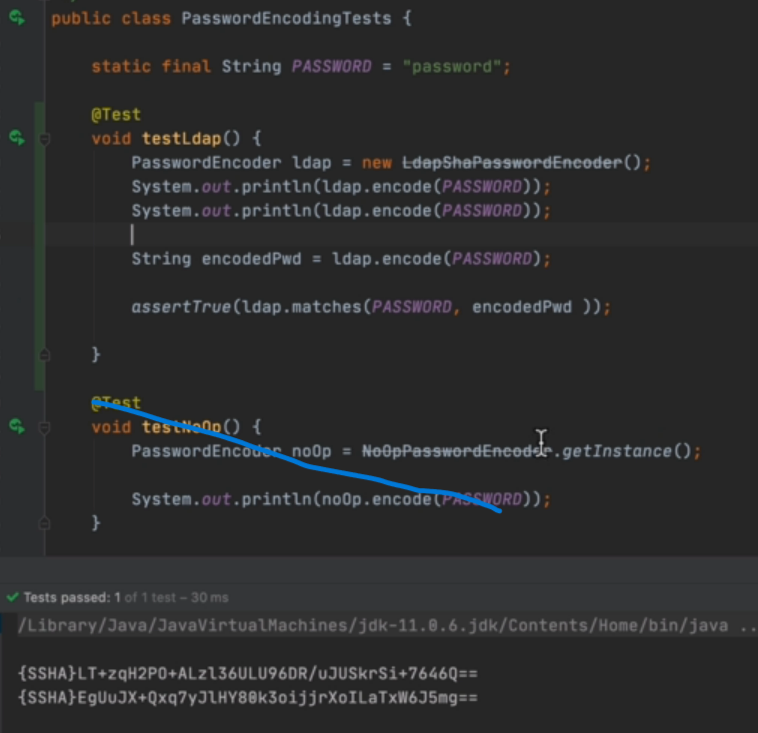


NoOp password Encoder: (not recommended)

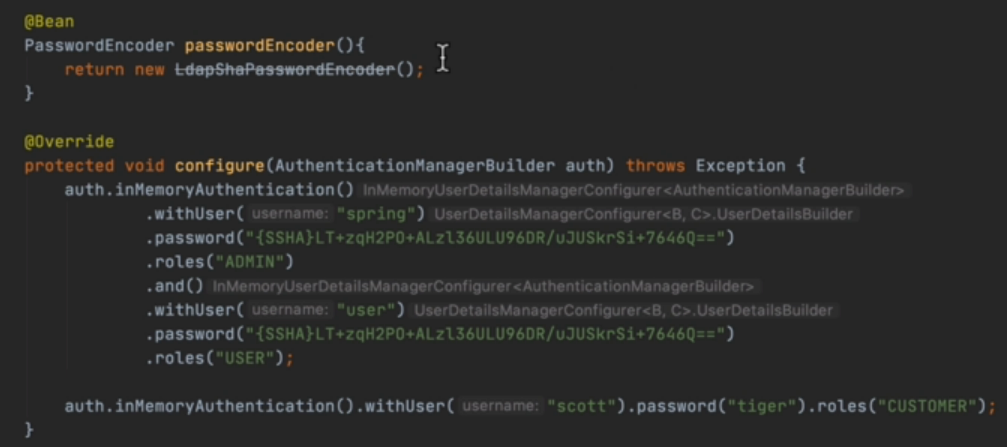


Ldap password Encoder: (not recommended)

-uses random salt



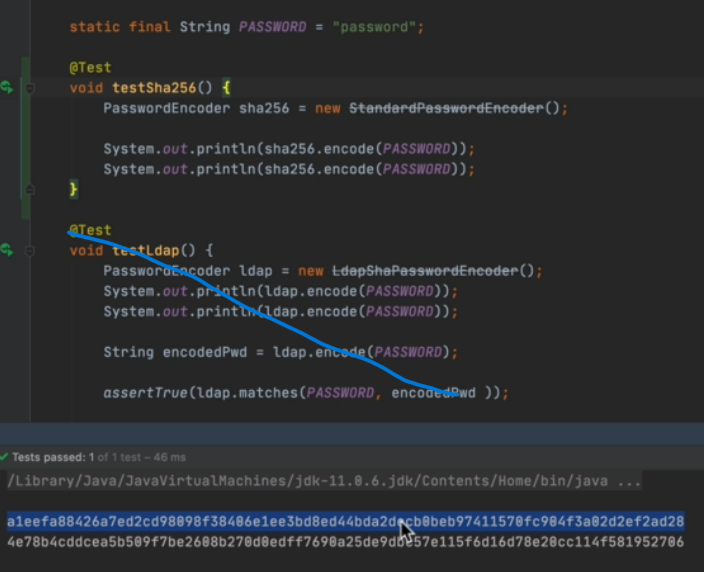
Declaring as Bean,



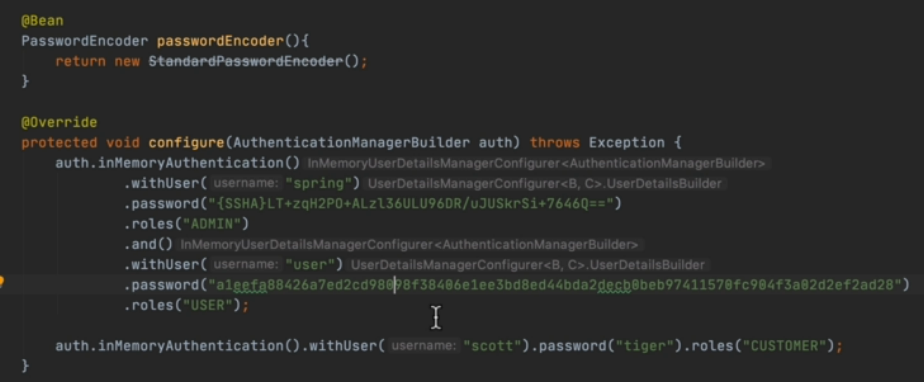
SHA-256 Password Encoder(not recommended)

-It was default in previous versions of spring security

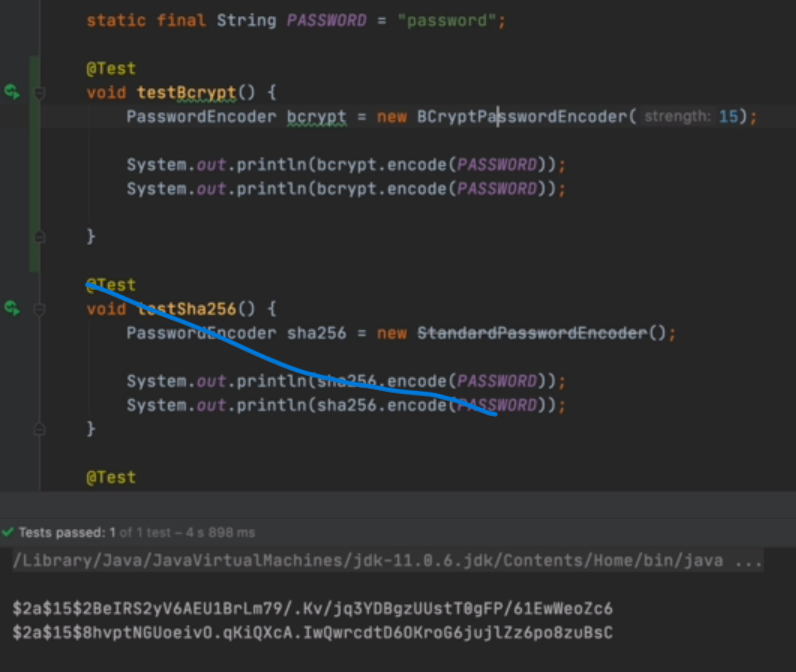
- Cons: tooFast in case of brut force attack. Brut Force attack requires more computational power. So other algorithms are slow compare to this.



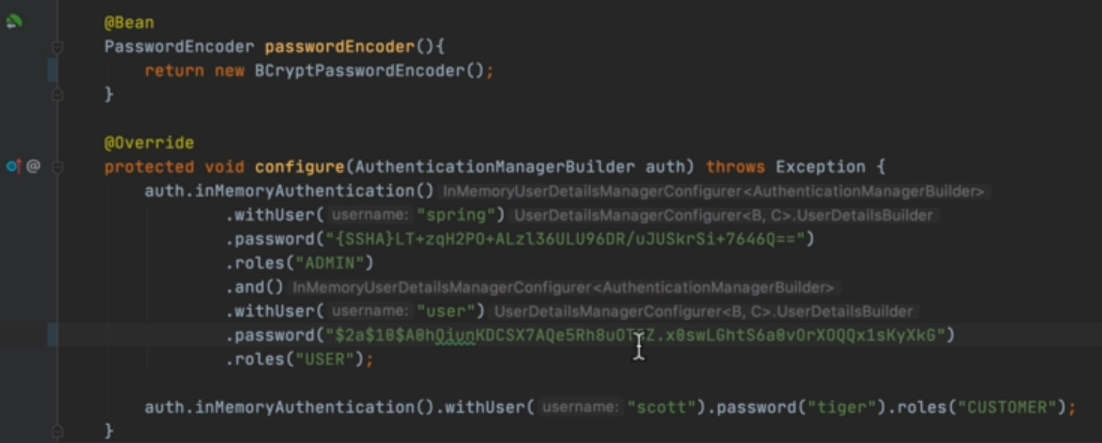
Declaring as Bean,



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)

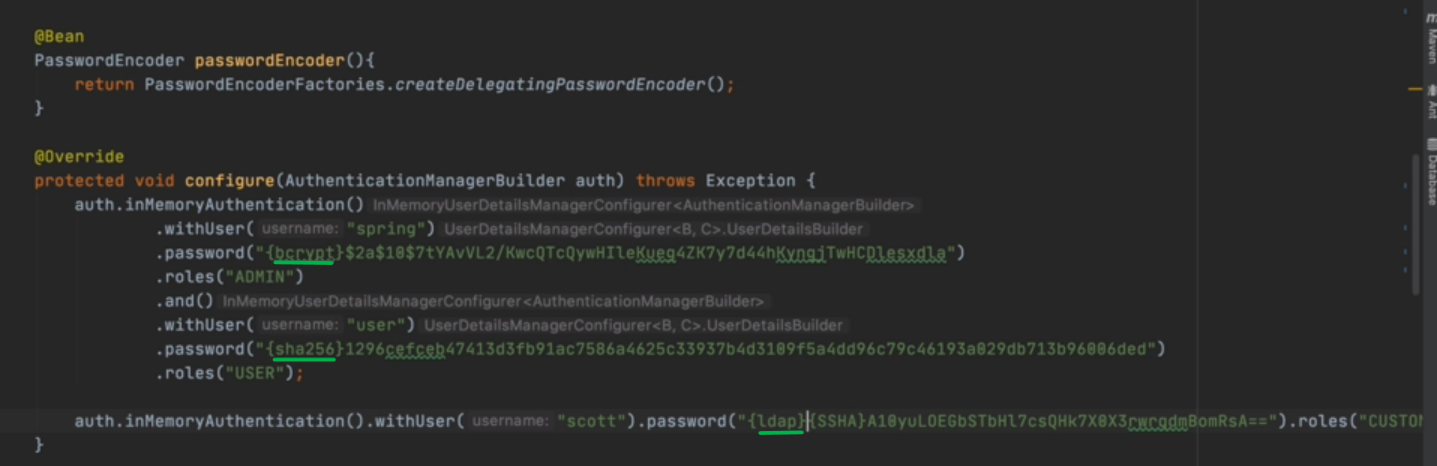


In testCase,



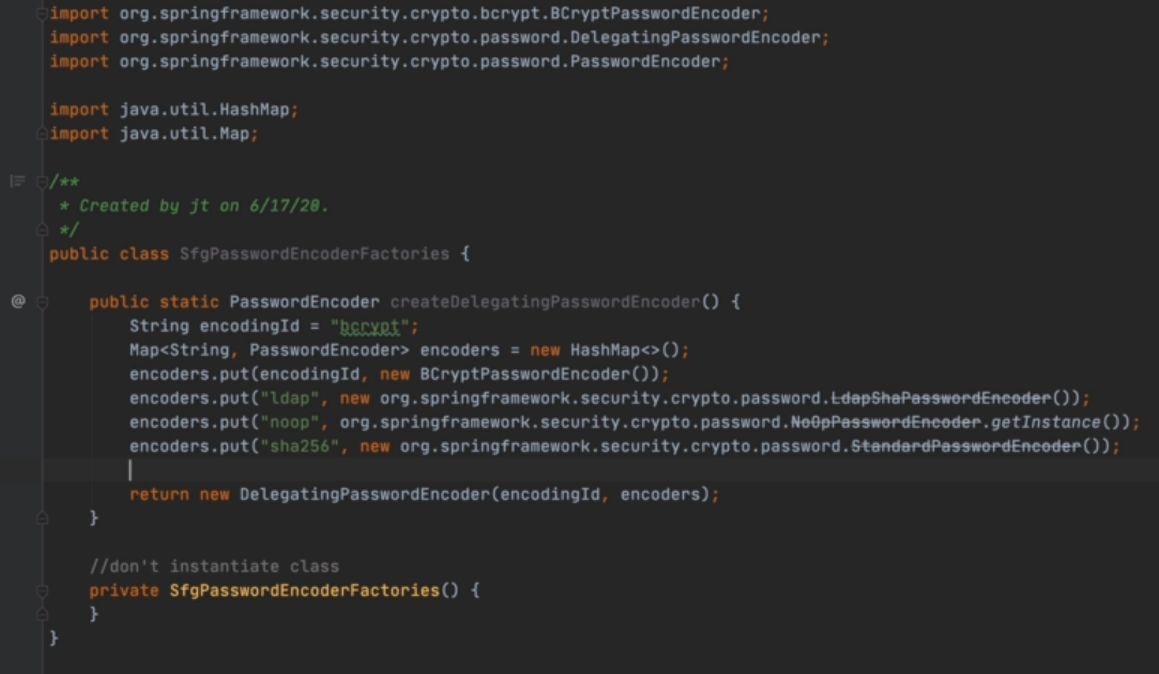
Delegating Password Encoder

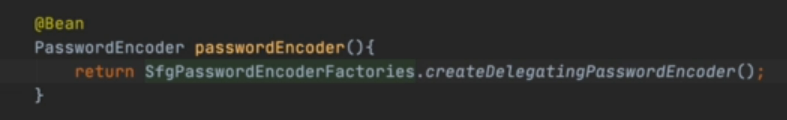
We can use multiple encoding algorithms at a time.



Custom Delegating Password Encoder

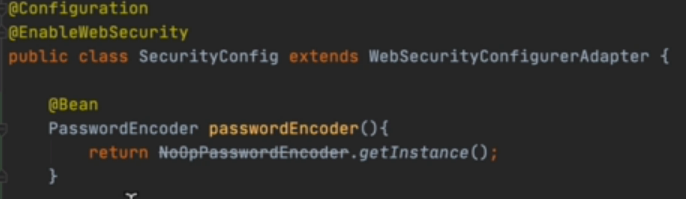
Create a class, here we are copying and paste the method “createDelegatingPasswordEncoder” from “PasswordEncoderFactoriesClass” and modify according to our need.





**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.

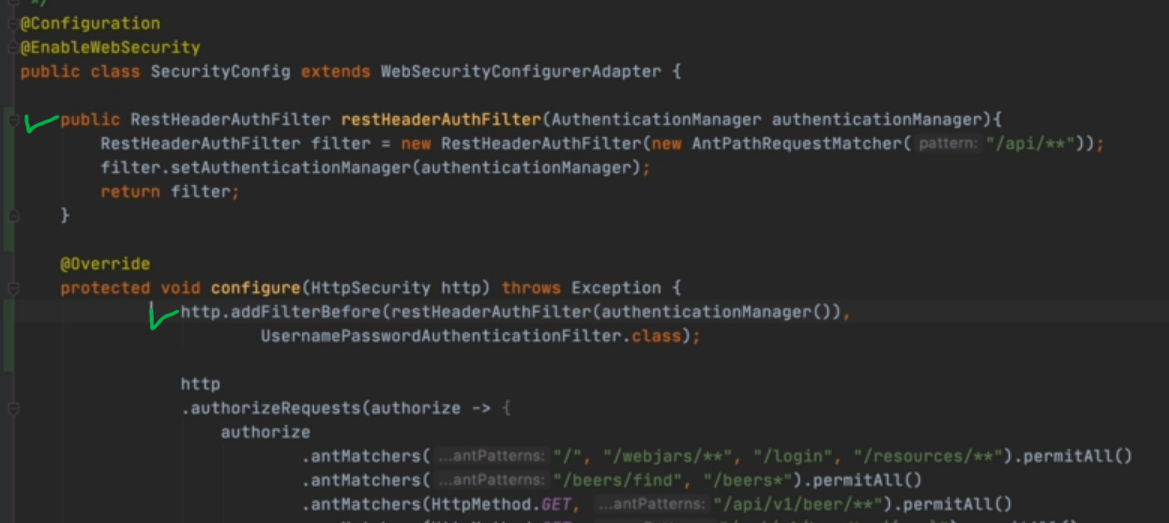
- If authentication is successful, the resulting [Authentication](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/Authentication.html) object will be placed into the SecurityContext for the current thread, which is guaranteed to have already been created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

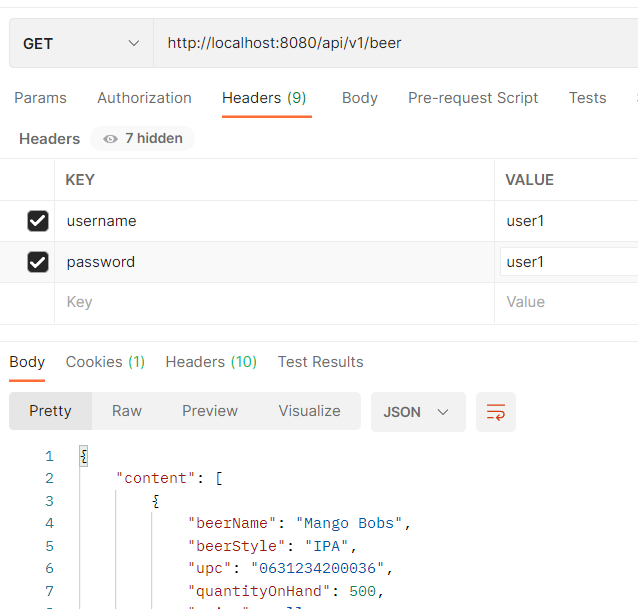
#Add this custom filter to security configuration

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

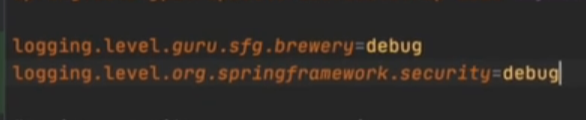


Img: filterConfig

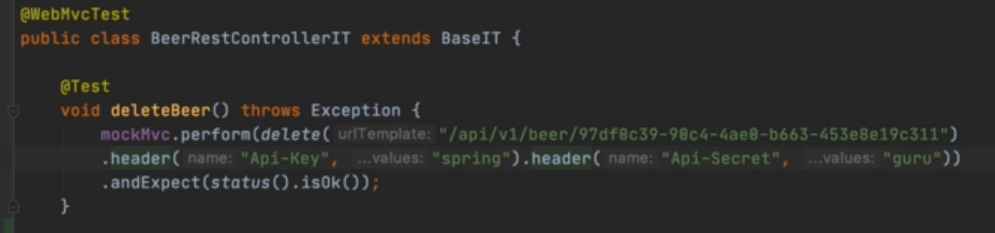
Observation: Hitting api in postman, we have added username and password in the header(in RestHeaderAuthFilter we are checking the headers for username and password). So based on the Authentication technique whether its httpbasic or custom userDetailsService …etc, it will fetch the username and password.



*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;  
 }

“””””””””

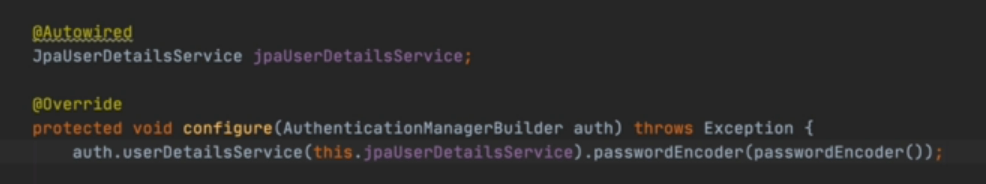
“””””””””

“”””””””” (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<>();  
 }  
 }  
}

Observation: As we annotated this class with @Service, spring will come to know about this userDetailsService config component, so no need to mention this class in SecurityConfig.java (below img). If we have multiple type of authentication we can use this technique (below img).



7. USER ROLE

Intro: [pdfs\25783944-AuthorizationInSpringSec.pdf](pdfs/25783944-AuthorizationInSpringSec.pdf) \*\*

**Note:**

-User Role should start with “ROLE\_” while saving in entity. But in security config we should use it without “ROLE\_”.

-User Authorities can ba any String

#Set a Specific role for deleting a beer and breweries

So here except ADMIN Role, no one can delete user, it is forbidden.

ADMIN & CUSTOMER can access breweries

package guru.sfg.brewery.config;  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 public RestHeaderAuthFilter restHeaderAuthFilter(AuthenticationManager authenticationManager){  
 RestHeaderAuthFilter filter = new RestHeaderAuthFilter(new AntPathRequestMatcher("/api/\*\*"));  
 filter.setAuthenticationManager(authenticationManager);  
 return filter;  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.addFilterBefore(restHeaderAuthFilter(authenticationManager()), UsernamePasswordAuthenticationFilter.class);  
 http  
 .authorizeRequests(authorize->{  
 authorize  
 .antMatchers(("/h2-console/\*\*")).permitAll() //not recomemded in production  
 .antMatchers("/","/webjars/\*\*","/login","/resources/\*\*").permitAll()  
 .antMatchers("/beers/find","/beers").permitAll()  
 .antMatchers(HttpMethod.*GET*, "/api/v1/beer/\*\*").permitAll()  
 .mvcMatchers(HttpMethod.*DELETE*, "/api/v1/beer/\*\*").hasRole("ADMIN")  
 .mvcMatchers(HttpMethod.*GET*, "/api/v1/beerUpc/{upc}").permitAll();

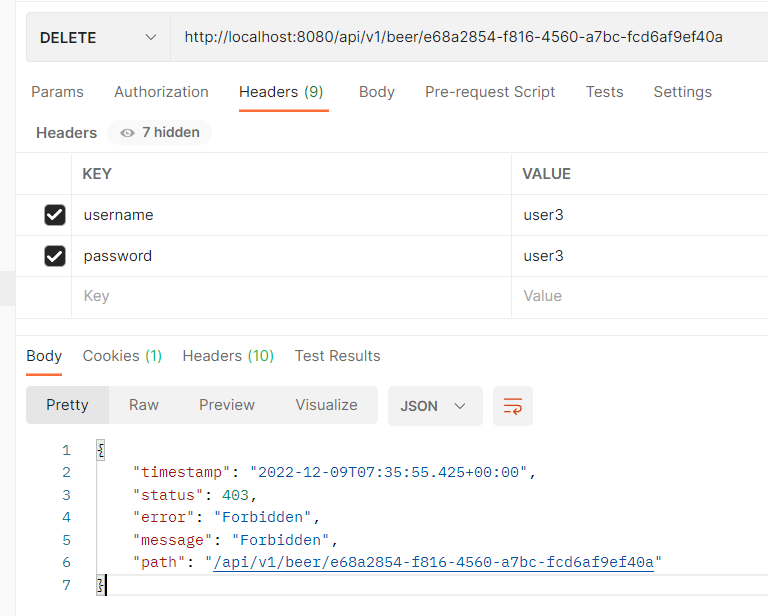
.mvcMatchers("/brewery/breweries").hasAnyRole("ADMIN","CUSTOMER")  
 .mvcMatchers(HttpMethod.*GET*, "brewery/api/v1/breweries").hasAnyRole("ADMIN","CUSTOMER");

})  
 .authorizeRequests()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .and()  
 .csrf().disable()  
 .httpBasic();  
  
 //h2 console config  
 http.headers().frameOptions().disable();  
 }

@Bean  
 PasswordEncoder passwordEncoder(){  
 return PasswordEncoderFactories.*createDelegatingPasswordEncoder*();  
 }}

Observation: hit delete api with user3 (role=”USER”), we will get forbidden error. User3 is authenticated but delete operation is forbideen.

(user1= ADMIN, user2=CUSTOMER, user3=USER) (refer to bootstrap class)



7.7 Adding Role Security at method level

#add @EnableGlobalMethodSecurity

@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(securedEnabled = true) //securedEnabled = Determines if Spring Security's Secured annotations should be enabled  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 public RestHeaderAuthFilter restHeaderAuthFilter(AuthenticationManager authenticationManager){  
 RestHeaderAuthFilter filter = new RestHeaderAuthFilter(new AntPathRequestMatcher("/api/\*\*"));  
 filter.setAuthenticationManager(authenticationManager);  
 return filter;  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.addFilterBefore(restHeaderAuthFilter(authenticationManager()), UsernamePasswordAuthenticationFilter.class);  
  
 http  
 .authorizeRequests(authorize->{  
 authorize  
 .antMatchers(("/h2-console/\*\*")).permitAll() //not recomemded in production  
 .antMatchers("/","/webjars/\*\*","/login","/resources/\*\*").permitAll()  
 .antMatchers("/beers/find","/beers").permitAll()  
 .antMatchers(HttpMethod.*GET*, "/api/v1/beer/\*\*").permitAll()  
 .mvcMatchers(HttpMethod.*DELETE*, "/api/v1/beer/\*\*").hasRole("ADMIN")  
 .mvcMatchers(HttpMethod.*GET*, "/api/v1/beerUpc/{upc}").permitAll()  
 .mvcMatchers("/brewery/breweries").hasAnyRole("ADMIN","CUSTOMER")  
 .mvcMatchers(HttpMethod.*GET*, "brewery/api/v1/breweries").hasAnyRole("ADMIN","CUSTOMER");  
 })  
 .authorizeRequests()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .and()  
 .csrf().disable()  
 .httpBasic();  
  
 //h2 console config  
 http.headers().frameOptions().disable();  
 }  
  
 @Bean  
 PasswordEncoder passwordEncoder(){  
 return PasswordEncoderFactories.*createDelegatingPasswordEncoder*();  
 }  
}

#add @Secure annotation at the method with allowed Roles.

@RequiredArgsConstructor  
@RequestMapping("/customers")  
@Controller  
public class CustomerController {

-----

@Secured({"ROLE\_ADMIN","ROLE\_CUSTOMER"})  
@GetMapping  
public String processFindFormReturnMany(Customer customer, BindingResult result, Model model){  
 // find customers by name  
 //*ToDO: Add Service* List<Customer> customers = customerRepository.findAllByCustomerNameLike("%" + customer.getCustomerName() + "%");  
 if (customers.isEmpty()) {  
 // no customers found  
 result.rejectValue("customerName", "notFound", "not found");  
 return "customers/findCustomers";  
 } else if (customers.size() == 1) {  
 // 1 customer found  
 customer = customers.get(0);  
 return "redirect:/customers/" + customer.getId();  
 } else {  
 // multiple customers found  
 model.addAttribute("selections", customers);  
 return "customers/customerList";  
 }  
}

7.71 Adding Role Security at method level using Spring Expression language

@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)   
public class SecurityConfig extends WebSecurityConfigurerAdapter {

----

Img: SecurityconfigSEL

method in

@RequiredArgsConstructor  
@RequestMapping("/customers")  
@Controller  
public class CustomerController {

-------

@PreAuthorize("hasRole('ADMIN')") //we can use hasAnyRole,.. also  
@PostMapping("/new")  
public String processCreationForm(Customer customer) {  
 //*ToDO: Add Service* Customer newCustomer = Customer.*builder*()  
 .customerName(customer.getCustomerName())  
 .build();  
  
 Customer savedCustomer= customerRepository.save(newCustomer);  
 return "redirect:/customers/" + savedCustomer.getId();  
}

8. USER authorities

8.1 Refactor Entities, Add Role entity and add RoleRepository

[pdfs\26063020-RefactorRoles.pdf](pdfs/26063020-RefactorRoles.pdf) \* (refer image of entity mapping)

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 role  
 @ManyToMany(cascade = {CascadeType.*MERGE*, CascadeType.*PERSIST*}, fetch = FetchType.*EAGER*)  
 @JoinTable(name="user\_role",  
 joinColumns = {@JoinColumn(name = "USER\_ID", referencedColumnName = "ID")},  
 inverseJoinColumns = {@JoinColumn(name = "ROLE\_ID", referencedColumnName = "ID")})  
 private Set<Role> roles;  
  
 @Transient  
 private Set<Authority> authorities;  
  
 public Set<Authority> getAuthorities() {  
 return this.roles.stream()  
 .map(Role::getAuthorities)  
 .flatMap(Set::stream)  
 .collect(Collectors.*toSet*());  
 }

----------------------------------------------

}

Role.js:

package guru.sfg.brewery.domain.security;

@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Entity  
public class Role {  
  
 @Id  
 @Column(name = "id", nullable = false)  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
  
 private String name;  
  
 @ManyToMany(mappedBy = "roles")  
 private Set<User> users;  
  
 @Singular //provide a singular method for adding an role  
 @ManyToMany(cascade = {CascadeType.*MERGE*, CascadeType.*PERSIST*}, fetch = FetchType.*EAGER*)  
 @JoinTable(name="role\_authority",  
 joinColumns = {@JoinColumn(name = "ROLE\_ID", referencedColumnName = "ID")},  
 inverseJoinColumns = {@JoinColumn(name = "AUTHORITY\_ID", referencedColumnName = "ID")})  
 private Set<Authority> authorities;  
  
}

Authority.js:

@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Entity  
public class Authority {  
  
 @Id  
 @Column(name = "id", nullable = false)  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
  
 private String permission;  
  
 @ManyToMany(mappedBy = "authorities")  
 private Set<Role> roles;  
}

RoleRepository.java:

public interface RoleRepository extends JpaRepository<Role, Long> {  
}

8.4 Bootstrap class. Data configuration to db.

package guru.sfg.brewery.bootstrap;  
@RequiredArgsConstructor  
@Component  
public class DefaultBreweryLoader implements CommandLineRunner {

------  
  
 public void loadUserAndAuthorityData(){  
  
 //Authorities  
 Authority createBeer = authorityRepository.save(Authority.*builder*().permission("beer.create").build());  
 Authority readBeer = authorityRepository.save(Authority.*builder*().permission("beer.read").build());  
 Authority updateBeer = authorityRepository.save(Authority.*builder*().permission("beer.update").build());  
 Authority deleteBeer = authorityRepository.save(Authority.*builder*().permission("beer.delete").build());

//Role   
 Role adminRole = roleRepository.save(Role.*builder*()  
 .name("ADMIN")  
 .authorities(Set.*of*(createBeer, readBeer, updateBeer, deleteBeer))  
 .build());  
 Role customerRole = roleRepository.save(Role.*builder*()  
 .name("CUSTOMER")  
 .authorities(Set.*of*(readBeer))  
 .build());  
 Role userRole = roleRepository.save(Role.*builder*()  
 .name("USER")  
 .authorities(Set.*of*(readBeer))  
 .build());

//User   
 User user1 = User.*builder*()  
 .username("user1")  
 .password(passwordEncoder.encode("user1"))  
 .role(adminRole) //here we are setting single entity instead of a Set<Authority>  
 .build();  
  
 User user2 = User.*builder*()  
 .username("user2")  
 .password(passwordEncoder.encode("user2"))  
 .role(customerRole)  
 .build();  
  
 User user3 = User.*builder*()  
 .username("user3")  
 .password(passwordEncoder.encode("user3"))  
 .role(userRole)  
 .build();  
  
 userRepository.save(user1);  
 userRepository.save(user2);  
 userRepository.save(user3);  
  
 System.*out*.println("saved successfully...");  
 }  
}

8.5 Refactoring hasRole() techniques in security config

If we see SecurityConfig we are using antMatchers, hasRole technique. When the application becomes large or complex, this technique will lead to complexity & errors. So to avoid this we will set authorities at the method level.

( Config remember: Img: SecurityconfigSEL )

#add @PreAuthorize to the controllers @preAuth -> [TLM]

package guru.sfg.brewery.web.controllers.api;   
@Slf4j  
@RequiredArgsConstructor  
@RequestMapping("/api/v1/")  
@RestController  
public class BeerRestController {  
 private static final Integer *DEFAULT\_PAGE\_NUMBER* = 0;  
 private static final Integer *DEFAULT\_PAGE\_SIZE* = 25;  
 private final BeerService beerService;  
  
 @PreAuthorize("hasAuthority('beer.read')")  
 @GetMapping(produces = { "application/json" }, path = "beer")  
 public ResponseEntity<BeerPagedList> listBeers(@RequestParam(value = "pageNumber", required = false) Integer pageNumber,  
 @RequestParam(value = "pageSize", required = false) Integer pageSize,  
 @RequestParam(value = "beerName", required = false) String beerName,  
 @RequestParam(value = "beerStyle", required = false) BeerStyleEnum beerStyle,  
 @RequestParam(value = "showInventoryOnHand", required = false) Boolean showInventoryOnHand){  
  
 *log*.debug("Listing Beers");  
  
 if (showInventoryOnHand == null) {  
 showInventoryOnHand = false;  
 }  
  
 if (pageNumber == null || pageNumber < 0){  
 pageNumber = *DEFAULT\_PAGE\_NUMBER*;  
 }  
  
 if (pageSize == null || pageSize < 1) {  
 pageSize = *DEFAULT\_PAGE\_SIZE*;  
 }  
  
 BeerPagedList beerList = beerService.listBeers(beerName, beerStyle, PageRequest.*of*(pageNumber, pageSize), showInventoryOnHand);  
  
 return new ResponseEntity<>(beerList, HttpStatus.*OK*);  
 }  
  
 @PreAuthorize("hasAuthority('beer.read')")  
 @GetMapping(path = {"beer/{beerId}"}, produces = { "application/json" })  
 public ResponseEntity<BeerDto> getBeerById(@PathVariable("beerId") UUID beerId,  
 @RequestParam(value = "showInventoryOnHand", required = false) Boolean showInventoryOnHand){  
  
 *log*.debug("Get Request for BeerId: " + beerId);  
  
 if (showInventoryOnHand == null) {  
 showInventoryOnHand = false;  
 }  
  
 return new ResponseEntity<>(beerService.findBeerById(beerId, showInventoryOnHand), HttpStatus.*OK*);  
 }  
  
 @PreAuthorize("hasAuthority('beer.read')")  
 @GetMapping(path = {"beerUpc/{upc}"}, produces = { "application/json" })  
 public ResponseEntity<BeerDto> getBeerByUpc(@PathVariable("upc") String upc){  
 return new ResponseEntity<>(beerService.findBeerByUpc(upc), HttpStatus.*OK*);  
 }  
  
 @PreAuthorize("hasAuthority('beer.create')")  
 @PostMapping(path = "beer")  
 public ResponseEntity saveNewBeer(@Valid @RequestBody BeerDto beerDto){  
  
 BeerDto savedDto = beerService.saveBeer(beerDto);  
  
 HttpHeaders httpHeaders = new HttpHeaders();  
  
 //*todo hostname for uri* httpHeaders.add("Location", "/api/v1/beer\_service/" + savedDto.getId().toString());  
  
 return new ResponseEntity(httpHeaders, HttpStatus.*CREATED*);  
 }  
  
 @PreAuthorize("hasAuthority('beer.update')")  
 @PutMapping(path = {"beer/{beerId}"}, produces = { "application/json" })  
 public ResponseEntity updateBeer(@PathVariable("beerId") UUID beerId, @Valid @RequestBody BeerDto beerDto){  
  
 beerService.updateBeer(beerId, beerDto);  
  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
  
 @PreAuthorize("hasAuthority('beer.delete')")  
 @DeleteMapping({"beer/{beerId}"})  
 @ResponseStatus(HttpStatus.*NO\_CONTENT*)  
 public void deleteBeer(@PathVariable("beerId") UUID beerId){  
 beerService.deleteById(beerId);  
 }  
  
 @ExceptionHandler(ConstraintViolationException.class)  
 @ResponseStatus(HttpStatus.*BAD\_REQUEST*)  
 ResponseEntity<List> badReqeustHandler(ConstraintViolationException e){  
 List<String> errors = new ArrayList<>(e.getConstraintViolations().size());  
  
 e.getConstraintViolations().forEach(constraintViolation -> {  
 errors.add(constraintViolation.getPropertyPath().toString() + " : " + constraintViolation.getMessage());  
 });  
  
 return new ResponseEntity<>(errors, HttpStatus.*BAD\_REQUEST*);  
 }}

Do the same in CustomerController.java and BeerController.java

#Refactor securityConfig (remove already configured code antMatchers, hasRole)

package guru.sfg.brewery.config;  
  
import guru.sfg.brewery.filters.RestHeaderAuthFilter;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.http.HttpMethod;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.crypto.factory.PasswordEncoderFactories;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;  
  
@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true) //securedEnabled = Determines if Spring Security's Secured annotations should be enabled  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 public RestHeaderAuthFilter restHeaderAuthFilter(AuthenticationManager authenticationManager){  
 RestHeaderAuthFilter filter = new RestHeaderAuthFilter(new AntPathRequestMatcher("/api/\*\*"));  
 filter.setAuthenticationManager(authenticationManager);  
 return filter;  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.addFilterBefore(restHeaderAuthFilter(authenticationManager()), UsernamePasswordAuthenticationFilter.class);  
  
 http  
 .authorizeRequests(authorize->{  
 authorize  
 .antMatchers(("/h2-console/\*\*")).permitAll() //not recomemded in production  
 .antMatchers("/","/webjars/\*\*","/login","/resources/\*\*").permitAll()

----------------------------------------------  
// .antMatchers("/beers/find","/beers").permitAll()  
// .antMatchers(HttpMethod.GET, "/api/v1/beer/\*\*").permitAll()  
// .mvcMatchers(HttpMethod.DELETE, "/api/v1/beer/\*\*").hasRole("ADMIN")  
// .mvcMatchers(HttpMethod.GET, "/api/v1/beerUpc/{upc}").permitAll()  
// .mvcMatchers("/brewery/breweries").hasAnyRole("ADMIN","CUSTOMER")  
// .mvcMatchers(HttpMethod.GET, "brewery/api/v1/breweries").hasAnyRole("ADMIN","CUSTOMER");

----------------------------------------------  
 })  
 .authorizeRequests()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .and()  
 .csrf().disable()  
 .httpBasic();  
  
 //h2 console config  
 http.headers().frameOptions().disable();  
 }  
  
 @Bean  
 PasswordEncoder passwordEncoder(){  
 return PasswordEncoderFactories.*createDelegatingPasswordEncoder*();  
 }  
}

# [TLM] -> @preAuth

if we see here we can see @PreAuthorize("hasAuthority('beer.read')") is repeated multiple time. To avoid this we can create a custom Annotation for ‘beer.create, beer.read, beer.update, beer.delete’.

@BeerCreatePermission

import org.springframework.security.access.prepost.PreAuthorize;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.create')")  
public @interface BeerCreatePermission {  
}

@BeerReadPermission

import org.springframework.security.access.prepost.PreAuthorize;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.read')")  
public @interface BeerReadPermission {  
}

@BeerUpdatePermission

import org.springframework.security.access.prepost.PreAuthorize;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.read')")  
public @interface BeerReadPermission {  
}

@BeerDeletePermission

import org.springframework.security.access.prepost.PreAuthorize;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.delete')")  
public @interface BeerDeletePermission {  
}

# Add above Annotations to required class (BeerRestController, CustomerController, BeerController)

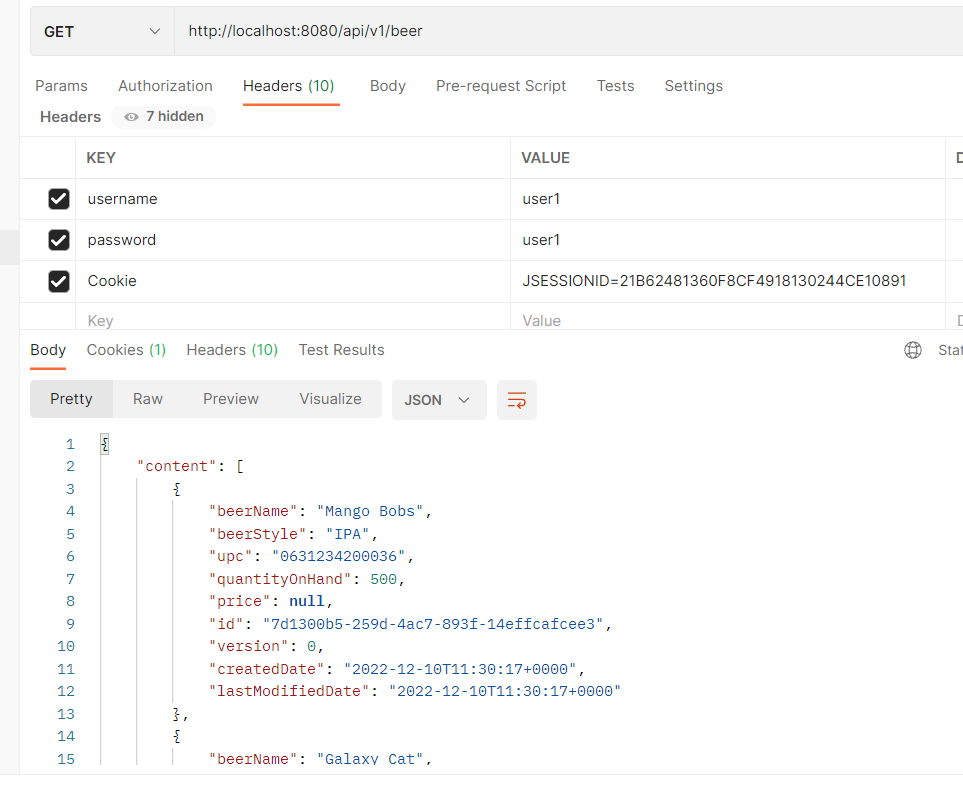
@Slf4j  
@RequiredArgsConstructor  
@RequestMapping("/api/v1/")  
@RestController  
public class BeerRestController {  
  
 private static final Integer *DEFAULT\_PAGE\_NUMBER* = 0;  
 private static final Integer *DEFAULT\_PAGE\_SIZE* = 25;  
  
 private final BeerService beerService;  
  
 @BeerReadPermission  
 @GetMapping(produces = { "application/json" }, path = "beer")  
 public ResponseEntity<BeerPagedList> listBeers(@RequestParam(value = "pageNumber", required = false) Integer pageNumber,  
 @RequestParam(value = "pageSize", required = false) Integer pageSize,  
 @RequestParam(value = "beerName", required = false) String beerName,  
 @RequestParam(value = "beerStyle", required = false) BeerStyleEnum beerStyle,  
 @RequestParam(value = "showInventoryOnHand", required = false) Boolean showInventoryOnHand){  
  
 *log*.debug("Listing Beers");  
  
 if (showInventoryOnHand == null) {  
 showInventoryOnHand = false;  
 }  
  
 if (pageNumber == null || pageNumber < 0){  
 pageNumber = *DEFAULT\_PAGE\_NUMBER*;  
 }  
  
 if (pageSize == null || pageSize < 1) {  
 pageSize = *DEFAULT\_PAGE\_SIZE*;  
 }  
 BeerPagedList beerList = beerService.listBeers(beerName, beerStyle, PageRequest.*of*(pageNumber, pageSize), showInventoryOnHand);  
  
 return new ResponseEntity<>(beerList, HttpStatus.*OK*);  
 }  
  
 @BeerReadPermission  
 @GetMapping(path = {"beer/{beerId}"}, produces = { "application/json" })  
 public ResponseEntity<BeerDto> getBeerById(@PathVariable("beerId") UUID beerId,  
 @RequestParam(value = "showInventoryOnHand", required = false) Boolean showInventoryOnHand){  
  
 *log*.debug("Get Request for BeerId: " + beerId);  
  
 if (showInventoryOnHand == null) {  
 showInventoryOnHand = false;  
 }  
  
 return new ResponseEntity<>(beerService.findBeerById(beerId, showInventoryOnHand), HttpStatus.*OK*);  
 }  
  
 @BeerReadPermission  
 @GetMapping(path = {"beerUpc/{upc}"}, produces = { "application/json" })  
 public ResponseEntity<BeerDto> getBeerByUpc(@PathVariable("upc") String upc){  
 return new ResponseEntity<>(beerService.findBeerByUpc(upc), HttpStatus.*OK*);  
 }  
  
 @BeerCreatePermission  
 @PostMapping(path = "beer")  
 public ResponseEntity saveNewBeer(@Valid @RequestBody BeerDto beerDto){  
  
 BeerDto savedDto = beerService.saveBeer(beerDto);  
  
 HttpHeaders httpHeaders = new HttpHeaders();  
  
 //*todo hostname for uri* httpHeaders.add("Location", "/api/v1/beer\_service/" + savedDto.getId().toString());  
  
 return new ResponseEntity(httpHeaders, HttpStatus.*CREATED*);  
 }  
  
 @BeerUpdatePermission  
 @PutMapping(path = {"beer/{beerId}"}, produces = { "application/json" })  
 public ResponseEntity updateBeer(@PathVariable("beerId") UUID beerId, @Valid @RequestBody BeerDto beerDto){  
  
 beerService.updateBeer(beerId, beerDto);  
  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
  
 @BeerDeletePermission  
 @DeleteMapping({"beer/{beerId}"})  
 @ResponseStatus(HttpStatus.*NO\_CONTENT*)  
 public void deleteBeer(@PathVariable("beerId") UUID beerId){  
 beerService.deleteById(beerId);  
 }  
  
 @ExceptionHandler(ConstraintViolationException.class)  
 @ResponseStatus(HttpStatus.*BAD\_REQUEST*)  
 ResponseEntity<List> badReqeustHandler(ConstraintViolationException e){  
 List<String> errors = new ArrayList<>(e.getConstraintViolations().size());  
  
 e.getConstraintViolations().forEach(constraintViolation -> {  
 errors.add(constraintViolation.getPropertyPath().toString() + " : " + constraintViolation.getMessage());  
 });  
  
 return new ResponseEntity<>(errors, HttpStatus.*BAD\_REQUEST*);  
 }  
  
}

Observation: \*\*

- Once the User is authenticated in the backend, a session is assigned for the user in the SecurityContextHolder and a JSESSIONID will get stored in browser (http only Cookie). For subsequent request this JSESSIONID will be passed in the requestHeader. So this JSESSIONID cookie will verified in the backend and whether session already exist for this user. If so the request is success.

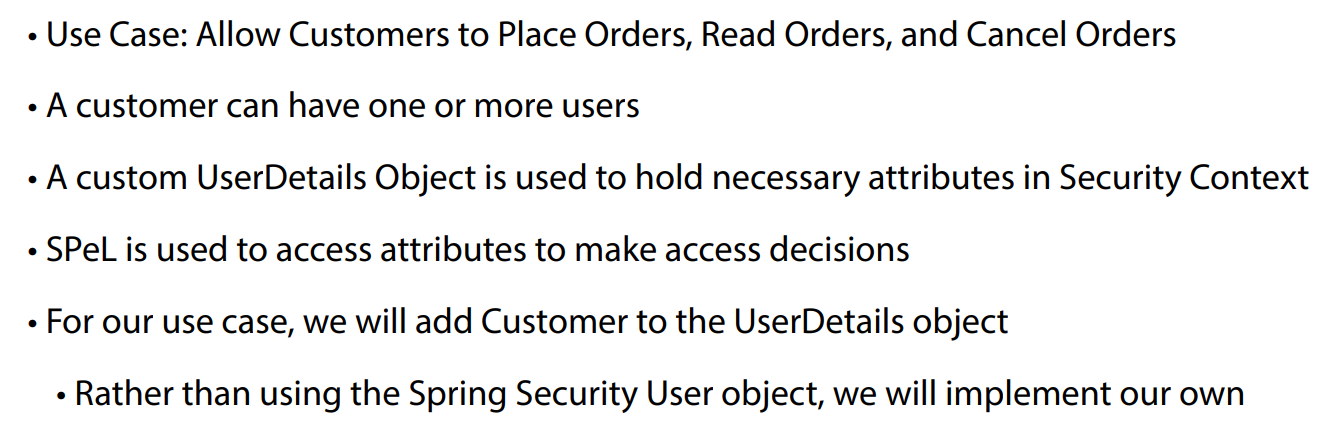
- Other filters can also access this SecurityContext, so no need of authentication again and again.

- below image we have given JSESSIONID in the header, without this we will get 401 error.



8. multitenancy security

Intro: [pdfs\26092220-MultitenancyOverview.pdf](pdfs/26092220-MultitenancyOverview.pdf) \*\*\*



#Add BeerOrderController

package guru.sfg.brewery.web.controllers.api;

@RequestMapping("/api/v1/customers/{customerId}/")  
@RestController  
public class BeerOrderController {  
 private static final Integer *DEFAULT\_PAGE\_NUMBER* = 0;  
 private static final Integer *DEFAULT\_PAGE\_SIZE* = 25;  
  
 private final BeerOrderService beerOrderService;  
  
 public BeerOrderController(BeerOrderService beerOrderService) {  
 this.beerOrderService = beerOrderService;  
 }  
  
 @GetMapping("orders")  
 public BeerOrderPagedList listOrders(@PathVariable("customerId") UUID customerId,  
 @RequestParam(value = "pageNumber", required = false) Integer pageNumber, @RequestParam(value = "pageSize", required = false) Integer pageSize) {  
 if (pageNumber == null || pageNumber < 0) {  
 pageNumber = *DEFAULT\_PAGE\_NUMBER*;  
 }  
  
 if (pageSize == null || pageSize < 1) {  
 pageSize = *DEFAULT\_PAGE\_SIZE*;  
 }  
 return beerOrderService.listOrders(customerId, PageRequest.*of*(pageNumber, pageSize));  
 }  
  
 @PostMapping("orders")  
 @ResponseStatus(HttpStatus.*CREATED*)  
 public BeerOrderDto placeOrder(@PathVariable("customerId") UUID customerId, @RequestBody BeerOrderDto beerOrderDto) {  
 return beerOrderService.placeOrder(customerId, beerOrderDto);  
 }  
  
 @GetMapping("orders/{orderId}")  
 public BeerOrderDto getOrder (@PathVariable("customerId") UUID customerId, @PathVariable("orderId") UUID orderId) { return beerOrderService.getOrderById(customerId, orderId);  
 }  
 @PutMapping("/orders/{orderId}/pickup") @ResponseStatus(HttpStatus.*NO\_CONTENT*)  
 public void pickupOrder (@PathVariable("customerId") UUID customerId, @PathVariable("orderId") UUID orderId) { beerOrderService.pickupOrder (customerId, orderId);  
 }  
}

#Configure bootstrap class for customer

@RequiredArgsConstructor  
@Component  
@Transactional  
public class DefaultBreweryLoader implements CommandLineRunner {  
  
 public static final String *TASTING\_ROOM* = "Tasting Room";  
 public static final String *BEER\_1\_UPC* = "0631234200036";  
 public static final String *BEER\_2\_UPC* = "0631234300019";  
 public static final String *BEER\_3\_UPC* = "0083783375213";  
  
 private final BreweryRepository breweryRepository;  
 private final BeerRepository beerRepository;  
 private final BeerInventoryRepository beerInventoryRepository;  
 private final BeerOrderRepository beerOrderRepository;  
 private final CustomerRepository customerRepository;  
 private final PasswordEncoder passwordEncoder;  
 private final UserRepository userRepository;  
 private final AuthorityRepository authorityRepository;  
 private final RoleRepository roleRepository;  
  
 @Override  
 public void run(String... args) {  
 loadBreweryData();  
 loadCustomerData();  
 loadUserAndAuthorityData();  
 }  
  
 private void loadCustomerData() {  
 Customer tastingRoom = Customer.*builder*()  
 .customerName(*TASTING\_ROOM*)  
 .apiKey(UUID.*randomUUID*())  
 .build();  
  
 customerRepository.save(tastingRoom);  
  
 beerRepository.findAll().forEach(beer -> {  
 beerOrderRepository.save(BeerOrder.*builder*()  
 .customer(tastingRoom)  
 .orderStatus(OrderStatusEnum.*NEW*)  
 .beerOrderLines(Set.*of*(BeerOrderLine.*builder*()  
 .beer(beer)  
 .orderQuantity(2)  
 .build()))  
 .build());  
 });  
 }  
  
 private void loadBreweryData() {  
 if (breweryRepository.count() == 0) {  
 breweryRepository.save(Brewery  
 .*builder*()  
 .breweryName("Cage Brewing")  
 .build());  
  
 Beer mangoBobs = Beer.*builder*()  
 .beerName("Mango Bobs")  
 .beerStyle(BeerStyleEnum.*IPA*)  
 .minOnHand(12)  
 .quantityToBrew(200)  
 .upc(*BEER\_1\_UPC*)  
 .build();  
  
 beerRepository.save(mangoBobs);  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beer(mangoBobs)  
 .quantityOnHand(500)  
 .build());  
  
 Beer galaxyCat = Beer.*builder*()  
 .beerName("Galaxy Cat")  
 .beerStyle(BeerStyleEnum.*PALE\_ALE*)  
 .minOnHand(12)  
 .quantityToBrew(200)  
 .upc(*BEER\_2\_UPC*)  
 .build();  
  
 beerRepository.save(galaxyCat);  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beer(galaxyCat)  
 .quantityOnHand(500)  
 .build());

Beer pinball = Beer.*builder*()  
 .beerName("Pinball Porter")  
 .beerStyle(BeerStyleEnum.*PORTER*)  
 .minOnHand(12)  
 .quantityToBrew(200)  
 .upc(*BEER\_3\_UPC*)  
 .build();  
  
 beerRepository.save(pinball);  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beer(pinball)  
 .quantityOnHand(500)  
 .build());  
  
 }  
 }

public void loadUserAndAuthorityData(){  
 //beer authorities  
 Authority createBeer = authorityRepository.save(Authority.*builder*().permission("beer.create").build());  
 Authority readBeer = authorityRepository.save(Authority.*builder*().permission("beer.read").build());  
 Authority updateBeer = authorityRepository.save(Authority.*builder*().permission("beer.update").build());  
 Authority deleteBeer = authorityRepository.save(Authority.*builder*().permission("beer.delete").build());

----------------------------------------------  
 //customer authorities  
 Authority createCustomer = authorityRepository.save(Authority.*builder*().permission("customer.create").build());  
 Authority readCustomer = authorityRepository.save(Authority.*builder*().permission("customer.read").build());  
 Authority updateCustomer = authorityRepository.save(Authority.*builder*().permission("customer.update").build());  
 Authority deleteCustomer = authorityRepository.save(Authority.*builder*().permission("customer.delete").build());  
  
 //brewery  
 Authority createBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.create").build());  
 Authority readBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.read").build());  
 Authority updateBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.update").build());  
 Authority deleteBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.delete").build());  
  
 //beer order customer  
 Authority createOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.create").build());  
 Authority readOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.read").build());  
 Authority updateOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.update").build());  
 Authority deleteOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.delete").build());  
  
 //beer order Admin  
 Authority createOrder = authorityRepository.save(Authority.*builder*().permission("order.create").build());  
 Authority readOrder = authorityRepository.save(Authority.*builder*().permission("order.read").build());  
 Authority updateOrder = authorityRepository.save(Authority.*builder*().permission("order.update").build());  
 Authority deleteOrder = authorityRepository.save(Authority.*builder*().permission("order.delete").build());  
----------------------------------------------  
 Role adminRole = roleRepository.save(Role.*builder*()  
 .name("ADMIN")  
 .authorities(Set.*of*(createBeer, readBeer, updateBeer, deleteBeer, createCustomer, readCustomer, updateCustomer, deleteCustomer, createBrewery, readBrewery, updateBrewery, deleteBrewery, createOrder, readOrder, updateOrder, deleteOrder))  
 .build());  
 Role customerRole = roleRepository.save(Role.*builder*()  
 .name("CUSTOMER")  
 .authorities(Set.*of*(readBeer, readCustomer, readBrewery, createOrderCustomer, readOrderCustomer, updateOrderCustomer, deleteOrderCustomer))  
 .build());  
 Role userRole = roleRepository.save(Role.*builder*()  
 .name("USER")  
 .authorities(Set.*of*(readBeer))  
 .build());  
  
 User user1 = User.*builder*()  
 .username("user1")  
 .password(passwordEncoder.encode("user1"))  
 .role(adminRole) //here we are setting single entity instead of a Set<Authority>  
 .build();  
  
 User user2 = User.*builder*()  
 .username("user2")  
 .password(passwordEncoder.encode("user2"))  
 .role(customerRole)  
 .build();  
  
 User user3 = User.*builder*()  
 .username("user3")  
 .password(passwordEncoder.encode("user3"))  
 .role(userRole)  
 .build();  
  
 userRepository.save(user1);  
 userRepository.save(user2);  
 userRepository.save(user3);  
  
 System.*out*.println("saved successfully...");  
 }  
}

# Update JPA User class

Implement with UserDetails and CredentialsContainer and override its methods. . ( we will be creating a map between User and Customer in the future, So if it was like before we cant achieve loadByUsername technique, because while returning in this method(loadByusername) we are using class **User** from the spring, which doesnot have Customer mapping).

Observation: here we are overriding getAuthorities() from the UserDetails interface

package guru.sfg.brewery.domain.security;  
  
@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Entity

public class User implements UserDetails, CredentialsContainer {  
 @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 role  
 @ManyToMany(cascade = {CascadeType.*MERGE*, CascadeType.*PERSIST*}, fetch = FetchType.*EAGER*)  
 @JoinTable(name="user\_role",  
 joinColumns = {@JoinColumn(name = "USER\_ID", referencedColumnName = "ID")},  
 inverseJoinColumns = {@JoinColumn(name = "ROLE\_ID", referencedColumnName = "ID")})  
 private Set<Role> roles;  
  
// @Transient  
// private Set<Authority> authorities;  
  
 @Transient  
 @Override  
 public Set<GrantedAuthority> getAuthorities() {  
 return this.roles.stream()  
 .map(Role::getAuthorities)  
 .flatMap(Set::stream)  
 .map(authority -> new SimpleGrantedAuthority(authority.getPermission()))  
 .collect(Collectors.*toSet*());  
 }  
----------------------------------------------  
 @Override  
 public boolean isAccountNonExpired() {  
 return this.accountNonExpired;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return this.accountNonLocked;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return this.credentialsNonExpired;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return this.enabled;  
 }  
  
 @Override  
 public void eraseCredentials() {  
 this.password = null;  
 }

----------------------------------------------  
}

#Update custom userDetailsService class

package guru.sfg.brewery.services.security;   
  
@Slf4j  
@RequiredArgsConstructor  
@Service  
public class JpaUserDetailsService implements UserDetailsService {  
  
 private final UserRepository userRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 *log*.debug("inside loadUserByUsername");  
  
 return userRepository.findByUsername(username).orElseThrow(()-> new UsernameNotFoundException("username "+username+" not found"));  
//before   
// 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::getPermission)  
// .map(SimpleGrantedAuthority::new)  
// .collect(Collectors.toList());  
// }else{  
// return new HashSet<>();  
// }  
// }  
}

# User & Customer class JPA mapping.

User:

package guru.sfg.brewery.domain.security;

@Getter  
@Setter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Entity  
public class User implements UserDetails, CredentialsContainer {  
 @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 role  
 @ManyToMany(cascade = {CascadeType.*MERGE*, CascadeType.*PERSIST*}, fetch = FetchType.*EAGER*)  
 @JoinTable(name="user\_role",  
 joinColumns = {@JoinColumn(name = "USER\_ID", referencedColumnName = "ID")},  
 inverseJoinColumns = {@JoinColumn(name = "ROLE\_ID", referencedColumnName = "ID")})  
 private Set<Role> roles;

----------------------------------------------  
 @ManyToOne(fetch = FetchType.*EAGER*)  
 private Customer customers;

----------------------------------------------  
  
// @Transient  
// private Set<Authority> authorities;  
  
 @Transient  
 @Override  
 public Set<GrantedAuthority> getAuthorities() {  
 return this.roles.stream()  
 .map(Role::getAuthorities)  
 .flatMap(Set::stream)  
 .map(authority -> new SimpleGrantedAuthority(authority.getPermission()))  
 .collect(Collectors.*toSet*());  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return this.accountNonExpired;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return this.accountNonLocked;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return this.credentialsNonExpired;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return this.enabled;  
 }  
  
 @Override  
 public void eraseCredentials() {  
 this.password = null;  
 }  
}

Customer:

package guru.sfg.brewery.domain;

@Getter  
@Setter  
@NoArgsConstructor  
@Entity  
public class Customer extends BaseEntity {  
  
 @Builder  
 public Customer(UUID id, Long version, Timestamp createdDate, Timestamp lastModifiedDate, String customerName,  
 UUID apiKey, Set<BeerOrder> beerOrders) {  
 super(id, version, createdDate, lastModifiedDate);  
 this.customerName = customerName;  
 this.apiKey = apiKey;  
 this.beerOrders = beerOrders;  
 }  
  
 private String customerName;  
  
 @Column(length = 36, columnDefinition = "varchar")  
 private UUID apiKey;  
  
 @OneToMany(mappedBy = "customer")  
 private Set<BeerOrder> beerOrders;

----------------------------------------------  
 @OneToMany(mappedBy = "customers", cascade = CascadeType.*ALL*, fetch = FetchType.*EAGER*)  
 private Set<User> users;

----------------------------------------------  
  
}

# Refactor bootstrap class

Adding 3 Customers and his 3 Users.

package guru.sfg.brewery.bootstrap;

@Slf4j  
@RequiredArgsConstructor  
@Component  
public class DefaultBreweryLoader implements CommandLineRunner {  
  
 public static final String *TASTING\_ROOM* = "Tasting Room";  
 public static final String *ST\_PETE\_DISTRIBUTING* = "St Pete distributing";  
 public static final String *DUNEDIN\_DISTRIBUTING* = "Dunedin distributing";  
 public static final String *KEY\_WEST\_DISTRIBUTORS* = "Key West Distributors";  
  
 public static final String *BEER\_1\_UPC* = "0631234200036";  
 public static final String *BEER\_2\_UPC* = "0631234300019";  
 public static final String *BEER\_3\_UPC* = "0083783375213";  
  
 private final BreweryRepository breweryRepository;  
 private final BeerRepository beerRepository;  
 private final BeerInventoryRepository beerInventoryRepository;  
 private final BeerOrderRepository beerOrderRepository;  
 private final CustomerRepository customerRepository;  
 private final PasswordEncoder passwordEncoder;  
 private final UserRepository userRepository;  
 private final AuthorityRepository authorityRepository;  
 private final RoleRepository roleRepository;

@Transactional  
 @Override  
 public void run(String... args) {  
 loadUserAndAuthorityData();  
 loadBreweryData();  
 loadTastingRoomData();  
 loadCustomerData();  
 }

----------------------------------------------  
 private void loadCustomerData() {  
 Role customerRole = roleRepository.findByName("CUSTOMER").orElseThrow(); //by default it throws NoSuchElementException("No value present")  
  
 //create customers  
 Customer stPeteCustomer = customerRepository.save(Customer.*builder*().customerName(*ST\_PETE\_DISTRIBUTING*)  
 .apiKey(UUID.*randomUUID*())  
 .build());  
 Customer dunedinCustomer = customerRepository.save(Customer.*builder*().customerName(*DUNEDIN\_DISTRIBUTING*).apiKey(UUID.*randomUUID*())  
 .build());  
 Customer keyWestCustomer = customerRepository.save(Customer.*builder*().customerName(*KEY\_WEST\_DISTRIBUTORS*)  
 .apiKey(UUID.*randomUUID*())  
 .build());  
  
 //create users (as of now we are not using this user data)  
 User stPeteUser = userRepository.save(User.*builder*().username("state")  
 .password(passwordEncoder.encode("password"))  
 .customer(stPeteCustomer)  
 .role(customerRole).build());  
 User dunedinUser = userRepository.save(User.*builder*().username("dunedin").password(passwordEncoder.encode("password"))  
 .customer(dunedinCustomer)  
 .role(customerRole).build());  
 User keywest = userRepository.save(User.*builder*().username("keywest").password(passwordEncoder.encode("password"))  
 .customer(keyWestCustomer)  
 .role(customerRole).build());  
  
 *log*.debug("Users loaded:"+userRepository.count()); //6   
  
 //create orders  
 createOrder(stPeteCustomer);  
 createOrder(dunedinCustomer);  
 createOrder(keyWestCustomer);  
  
 *log*.debug("Orders loaded:"+beerOrderRepository.count()); //6  
 }  
  
 private BeerOrder createOrder(Customer customer) {  
 return beerOrderRepository.save(BeerOrder.*builder*()  
 .customer(customer)  
 .orderStatus(OrderStatusEnum.*NEW*)  
 .beerOrderLines(Set.*of*(BeerOrderLine.*builder*()  
 .beer(beerRepository.findByUpc(*BEER\_1\_UPC*))  
 .orderQuantity(2)  
 .build()))  
 .build());  
 }  
----------------------------------------------

//just method name changed, before it was loadCustomerData  
 private void loadTastingRoomData() {   
 Customer tastingRoom = Customer.*builder*()  
 .customerName(*TASTING\_ROOM*)  
 .apiKey(UUID.*randomUUID*())  
 .build();  
  
 customerRepository.save(tastingRoom);  
  
 beerRepository.findAll().forEach(beer -> {  
 beerOrderRepository.save(BeerOrder.*builder*()  
 .customer(tastingRoom)  
 .orderStatus(OrderStatusEnum.*NEW*)  
 .beerOrderLines(Set.*of*(BeerOrderLine.*builder*()  
 .beer(beer)  
 .orderQuantity(2)  
 .build()))  
 .build());  
 });  
 }  
  
 private void loadBreweryData() {  
 if (breweryRepository.count() == 0) {  
 breweryRepository.save(Brewery  
 .*builder*()  
 .breweryName("Cage Brewing")  
 .build());  
  
 Beer mangoBobs = Beer.*builder*()  
 .beerName("Mango Bobs")  
 .beerStyle(BeerStyleEnum.*IPA*)  
 .minOnHand(12)  
 .quantityToBrew(200)  
 .upc(*BEER\_1\_UPC*)  
 .build();  
  
 beerRepository.save(mangoBobs);  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beer(mangoBobs)  
 .quantityOnHand(500)  
 .build());  
  
 Beer galaxyCat = Beer.*builder*()  
 .beerName("Galaxy Cat")  
 .beerStyle(BeerStyleEnum.*PALE\_ALE*)  
 .minOnHand(12)  
 .quantityToBrew(200)  
 .upc(*BEER\_2\_UPC*)  
 .build();  
  
 beerRepository.save(galaxyCat);  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beer(galaxyCat)  
 .quantityOnHand(500)  
 .build());  
  
 Beer pinball = Beer.*builder*()  
 .beerName("Pinball Porter")  
 .beerStyle(BeerStyleEnum.*PORTER*)  
 .minOnHand(12)  
 .quantityToBrew(200)  
 .upc(*BEER\_3\_UPC*)  
 .build();  
  
 beerRepository.save(pinball);  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beer(pinball)  
 .quantityOnHand(500)  
 .build());  
  
 }  
 }  
  
 public void loadUserAndAuthorityData() {  
 //beer authorities  
 Authority createBeer = authorityRepository.save(Authority.*builder*().permission("beer.create").build());  
 Authority readBeer = authorityRepository.save(Authority.*builder*().permission("beer.read").build());  
 Authority updateBeer = authorityRepository.save(Authority.*builder*().permission("beer.update").build());  
 Authority deleteBeer = authorityRepository.save(Authority.*builder*().permission("beer.delete").build());  
  
 //customer authorities  
 Authority createCustomer = authorityRepository.save(Authority.*builder*().permission("customer.create").build());  
 Authority readCustomer = authorityRepository.save(Authority.*builder*().permission("customer.read").build());  
 Authority updateCustomer = authorityRepository.save(Authority.*builder*().permission("customer.update").build());  
 Authority deleteCustomer = authorityRepository.save(Authority.*builder*().permission("customer.delete").build());  
  
 //brewery  
 Authority createBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.create").build());  
 Authority readBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.read").build());  
 Authority updateBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.update").build());  
 Authority deleteBrewery = authorityRepository.save(Authority.*builder*().permission("brewery.delete").build());  
  
 //beer order customer  
 Authority createOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.create").build());  
 Authority readOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.read").build());  
 Authority updateOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.update").build());  
 Authority deleteOrderCustomer = authorityRepository.save(Authority.*builder*().permission("customer.order.delete").build());  
  
 // beer order Admin  
 Authority createOrder = authorityRepository.save(Authority.*builder*().permission("order.create").build());  
 Authority readOrder = authorityRepository.save(Authority.*builder*().permission("order.read").build());  
 Authority updateOrder = authorityRepository.save(Authority.*builder*().permission("order.update").build());  
 Authority deleteOrder = authorityRepository.save(Authority.*builder*().permission("order.delete").build());  
  
 Role adminRole = roleRepository.save(Role.*builder*()  
 .name("ADMIN")  
 .authorities(Set.*of*(createBeer, readBeer, updateBeer, deleteBeer, createCustomer, readCustomer, updateCustomer, deleteCustomer,  
 createBrewery, readBrewery, updateBrewery, deleteBrewery, createOrder, readOrder, updateOrder, deleteOrder))  
 .build());  
 Role customerRole = roleRepository.save(Role.*builder*()  
 .name("CUSTOMER")  
 .authorities(Set.*of*(readBeer, readCustomer, readBrewery, createOrderCustomer, readOrderCustomer, updateOrderCustomer, deleteOrderCustomer))  
 .build());  
 Role userRole = roleRepository.save(Role.*builder*()  
 .name("USER")  
 .authorities(Set.*of*(readBeer))  
 .build());  
  
 User user1 = User.*builder*()  
 .username("user1")  
 .password(passwordEncoder.encode("user1"))  
 .role(adminRole) //here we are setting single entity instead of a Set<Authority>  
 .build();  
  
 User user2 = User.*builder*()  
 .username("user2")  
 .password(passwordEncoder.encode("user2"))  
 .role(customerRole)  
 .build();  
  
 User user3 = User.*builder*()  
 .username("user3")  
 .password(passwordEncoder.encode("user3"))  
 .role(userRole)  
 .build();  
  
 userRepository.save(user1);  
 userRepository.save(user2);  
 userRepository.save(user3);  
  
 System.*out*.println("saved successfully...");  
 }  
}

8.8 Use Case [pdfs\26100804-Use-Case.pdf](pdfs/26100804-Use-Case.pdf) \*\*\*

#Set up Custom Authentication Manager

package guru.sfg.brewery.security;  
  
@Slf4j  
@Component  
public class BeerOrderAuthenticationManager {  
  
 public boolean customerIdMatches(Authentication authentication, UUID customerId){  
 User authenticationUser = (User) authentication.getPrincipal();  
  
 *log*.debug("Auth user customerId: "+authenticationUser.getCustomer().getId()+" customerId: "+customerId);  
  
 return customerId.equals(authenticationUser.getCustomer().getId());  
 }  
}

#Using Authentication principle in controller

( we are going to depricate this V1 bcz we are not supposed to show customerId in the url) (refer below for V2)

Using BeerOrderAuthenticationManager in the @PreAthorize

package guru.sfg.brewery.web.controllers.api;   
  
@RequestMapping("/api/v1/customers/{customerId}/")  
@RestController  
public class BeerOrderController {  
 private static final Integer *DEFAULT\_PAGE\_NUMBER* = 0;  
 private static final Integer *DEFAULT\_PAGE\_SIZE* = 25;  
 private final BeerOrderService beerOrderService;  
  
 public BeerOrderController(BeerOrderService beerOrderService) {  
 this.beerOrderService = beerOrderService;  
 }  
  
 @PreAuthorize("hasAuthority('order.read') OR " +  
 "hasAuthority('customer.order.read') AND " +  
 "@beerOrderAuthenticationManager.customerIdMatches(authentication, #customerId)")  
 @GetMapping("orders")  
 public BeerOrderPagedList listOrders(@PathVariable("customerId") UUID customerId,   
 @RequestParam(value = "pageNumber", required = false) Integer pageNumber, @RequestParam(value = "pageSize", required = false) Integer pageSize) {  
 if (pageNumber == null || pageNumber < 0) {  
 pageNumber = *DEFAULT\_PAGE\_NUMBER*;  
 }  
  
 if (pageSize == null || pageSize < 1) {  
 pageSize = *DEFAULT\_PAGE\_SIZE*;  
 }  
 return beerOrderService.listOrders(customerId, PageRequest.*of*(pageNumber, pageSize));  
 }  
  
 @PostMapping("orders")  
 @ResponseStatus(HttpStatus.*CREATED*)  
 public BeerOrderDto placeOrder(@PathVariable("customerId") UUID customerId, @RequestBody BeerOrderDto beerOrderDto) {  
 return beerOrderService.placeOrder(customerId, beerOrderDto);  
 }

@PreAuthorize("hasAuthority('order.read') OR " +  
 "hasAuthority('customer.order.read') AND " +  
 "@beerOrderAuthenticationManager.customerIdMatches(authentication, #customerId)")  
 @GetMapping("orders/{orderId}")  
 public BeerOrderDto getOrder (@PathVariable("customerId") UUID customerId, @PathVariable("orderId") UUID orderId) { return beerOrderService.getOrderById(customerId, orderId);  
 }

@PutMapping("/orders/{orderId}/pickup") @ResponseStatus(HttpStatus.*NO\_CONTENT*)  
 public void pickupOrder (@PathVariable("customerId") UUID customerId, @PathVariable("orderId") UUID orderId) { beerOrderService.pickupOrder (customerId, orderId);  
 }  
}

#Refactor BeerPermission Annotations

@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.create') OR hasAuthority('customer.order.create')")  
public @interface BeerCreatePermission {  
}

@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.read') OR hasAuthority('customer.order.read')")  
public @interface BeerReadPermission {  
}

@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.update') OR hasAuthority('customer.order.update')")  
public @interface BeerUpdatePermission {  
}

@Retention(RetentionPolicy.*RUNTIME*)  
@PreAuthorize("hasAuthority('beer.delete') OR hasAuthority('customer.order.delete')")  
public @interface BeerDeletePermission {  
}

**Using Spring Security with Spring Data Jpa \*\***

Advantage: we can use spring security techniques like hasRole, hasAuthority,principal… in the sql query. (quick refer Img:SWJ )

#dependency

<dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-data</artifactId>  
</dependency>

#Declare a bean in SecurityConfig

package guru.sfg.brewery.config;  
  
@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true) //securedEnabled = Determines if Spring Security's Secured annotations should be enabled  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
----------------------------------------------  
 // needed for use Spring Security with Spring Data JPA SPel  
 @Bean  
 public SecurityEvaluationContextExtension securityEvaluationContextExtension(){  
 return new SecurityEvaluationContextExtension();  
 }

@Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .authorizeRequests(authorize->{  
 authorize  
 .antMatchers(("/h2-console/\*\*")).permitAll() //not recomemded in production  
 .antMatchers("/","/webjars/\*\*","/login","/resources/\*\*").permitAll();  
 })  
 .authorizeRequests()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin()  
 .and()  
 .csrf().disable()  
 .httpBasic();  
  
 //h2 console config  
 http.headers().frameOptions().disable();  
 }  
 @Bean  
 PasswordEncoder passwordEncoder(){  
 return PasswordEncoderFactories.*createDelegatingPasswordEncoder*();  
 }  
}

# Create V2 for BeerOrderController

package guru.sfg.brewery.web.controllers.api;  
  
@RequestMapping("/api/v2/orders/")  
@RestController  
public class BeerOrderControllerV2 {  
 private static final Integer *DEFAULT\_PAGE\_NUMBER* = 0;  
 private static final Integer *DEFAULT\_PAGE\_SIZE* = 25;  
 private final BeerOrderService beerOrderService;  
  
 public BeerOrderControllerV2(BeerOrderService beerOrderService) {  
 this.beerOrderService = beerOrderService;  
 }  
  
 @BeerReadPermission  
 @GetMapping  
 public BeerOrderPagedList listOrders(@AuthenticationPrincipal User user, //removed customerId PathParam  
 @RequestParam(value = "pageNumber", required = false) Integer pageNumber,  
 @RequestParam(value = "pageSize", required = false) Integer pageSize) {  
 if (pageNumber == null || pageNumber < 0) {  
 pageNumber = *DEFAULT\_PAGE\_NUMBER*;  
 }  
  
 if (pageSize == null || pageSize < 1) {  
 pageSize = *DEFAULT\_PAGE\_SIZE*;  
 }  
 ----------------------------------------------  
 //If customer exist for the user fetch based on customerId or fetch All beerOrders.  
 // (only Admin(user1),Customer(user2),User(user3) don't have customer)  
 if(user.getCustomer()!=null){  
 return beerOrderService.listOrders(user.getCustomer().getId(), PageRequest.*of*(pageNumber, pageSize));  
 }else{  
 return beerOrderService.listOrders(PageRequest.*of*(pageNumber, pageSize));  
 }

----------------------------------------------  
 }  
  
 @PostMapping  
 @ResponseStatus(HttpStatus.*CREATED*)  
 public BeerOrderDto placeOrder(@PathVariable("customerId") UUID customerId, @RequestBody BeerOrderDto beerOrderDto) {  
 return beerOrderService.placeOrder(customerId, beerOrderDto);  
 }

@BeerReadPermission  
 @GetMapping("{orderId}")  
 public BeerOrderDto getOrder (@PathVariable("orderId") UUID orderId) { //removed customerId PathParam  
 ----------------------------------------------

BeerOrderDto beerOrderDto = beerOrderService.getOrderById(orderId);  
  
 if(beerOrderDto == null){  
 throw new ResponseStatusException(HttpStatus.*NOT\_FOUND*, "Order not found");  
 }  
 *log*.debug("Found Order: "+beerOrderDto);  
  
 return beerOrderDto;

----------------------------------------------  
 }  
  
 @PutMapping("{orderId}/pickup") @ResponseStatus(HttpStatus.*NO\_CONTENT*)  
 public void pickupOrder (@PathVariable("customerId") UUID customerId,  
 @PathVariable("orderId") UUID orderId) {  
 beerOrderService.pickupOrder (customerId, orderId);  
 }  
}

BeerOrderService.java (I)

public interface BeerOrderService {  
 BeerOrderPagedList listOrders(UUID customerId, Pageable pageable);  
 BeerOrderDto placeOrder(UUID customerId, BeerOrderDto beerOrderDto);  
 BeerOrderDto getOrderById(UUID customerId, UUID orderId);  
 void pickupOrder(UUID customerId, UUID orderId);  
 BeerOrderPagedList listOrders(Pageable pageable);  
  
 BeerOrderDto getOrderById(UUID orderId);  
}

BeerOrderServiceImpl.java

package guru.sfg.brewery.services;  
  
@Slf4j  
@Service  
@AllArgsConstructor  
public class BeerOrderServiceImpl implements BeerOrderService {  
  
 private final BeerOrderRepository beerOrderRepository;  
 private final CustomerRepository customerRepository;  
 private final BeerOrderMapper beerOrderMapper;  
  
 @Override  
 public BeerOrderPagedList listOrders(UUID customerId, Pageable pageable) {  
 Optional<Customer> customerOptional = customerRepository.findById(customerId);  
  
 if (customerOptional.isPresent()) {  
 Page<BeerOrder> beerOrderPage =  
 beerOrderRepository.findAllByCustomer(customerOptional.get(), pageable);  
  
 return new BeerOrderPagedList(beerOrderPage  
 .stream()  
 .map(beerOrderMapper::beerOrderToDto)  
 .collect(Collectors.*toList*()), PageRequest.*of*(  
 beerOrderPage.getPageable().getPageNumber(),  
 beerOrderPage.getPageable().getPageSize()),  
 beerOrderPage.getTotalElements());  
 } else {  
 return null;  
 }  
 }  
  
 @Override  
 public BeerOrderPagedList listOrders(Pageable pageable) {  
 Page<BeerOrder> beerOrderPage =  
 beerOrderRepository.findAll(pageable);  
  
 return new BeerOrderPagedList(beerOrderPage  
 .stream()  
 .map(beerOrderMapper::beerOrderToDto)  
 .collect(Collectors.*toList*()), PageRequest.*of*(  
 beerOrderPage.getPageable().getPageNumber(),  
 beerOrderPage.getPageable().getPageSize()),  
 beerOrderPage.getTotalElements());  
 }  
  
 @Transactional  
 @Override  
 public BeerOrderDto placeOrder(UUID customerId, BeerOrderDto beerOrderDto) {  
 Optional<Customer> customerOptional = customerRepository.findById(customerId);  
  
 if (customerOptional.isPresent()) {  
 BeerOrder beerOrder = beerOrderMapper.dtoToBeerOrder(beerOrderDto);  
 beerOrder.setId(null); //should not be set by outside client  
 beerOrder.setCustomer(customerOptional.get());  
 beerOrder.setOrderStatus(OrderStatusEnum.*NEW*);  
  
 beerOrder.getBeerOrderLines().forEach(line -> line.setBeerOrder(beerOrder));  
  
 BeerOrder savedBeerOrder = beerOrderRepository.saveAndFlush(beerOrder);  
  
 *log*.debug("Saved Beer Order: " + beerOrder.getId());  
  
 return beerOrderMapper.beerOrderToDto(savedBeerOrder);  
 }  
 //*todo add exception type* throw new RuntimeException("Customer Not Found");  
 }  
  
 @Override  
 public BeerOrderDto getOrderById(UUID customerId, UUID orderId) {  
 return beerOrderMapper.beerOrderToDto(getOrder(customerId, orderId));  
 }  
 ----------------------------------------------  
 @Override  
 public BeerOrderDto getOrderById(UUID orderId) {  
 BeerOrder beerOrder = beerOrderRepository.findOrderByIdSecure(orderId);  
 return beerOrderMapper.beerOrderToDto(beerOrder);  
 }

----------------------------------------------

@Override  
 public void pickupOrder(UUID customerId, UUID orderId) {  
 BeerOrder beerOrder = getOrder(customerId, orderId);  
 beerOrder.setOrderStatus(OrderStatusEnum.*PICKED\_UP*);  
  
 beerOrderRepository.save(beerOrder);  
 }

private BeerOrder getOrder(UUID customerId, UUID orderId){  
 Optional<Customer> customerOptional = customerRepository.findById(customerId);  
  
 if(customerOptional.isPresent()){  
 Optional<BeerOrder> beerOrderOptional = beerOrderRepository.findById(orderId);  
  
 if(beerOrderOptional.isPresent()){  
 BeerOrder beerOrder = beerOrderOptional.get();  
  
 // fall to exception if customer id's do not match - order not for customer  
 if(beerOrder.getCustomer().getId().equals(customerId)){  
 return beerOrder;  
 }  
 }  
 throw new RuntimeException("Beer Order Not Found");  
 }  
 throw new RuntimeException("Customer Not Found");  
 }  
}

BeerOrderRepository.java:

package guru.sfg.brewery.repositories;  
public interface BeerOrderRepository extends JpaRepository<BeerOrder, UUID> {  
  
 Page<BeerOrder> findAllByCustomer(Customer customer, Pageable pageable);  
  
 List<BeerOrder> findAllByOrderStatus(OrderStatusEnum orderStatusEnum);  
  
 @Lock(LockModeType.*PESSIMISTIC\_WRITE*)  
 BeerOrder findOneById(UUID id);

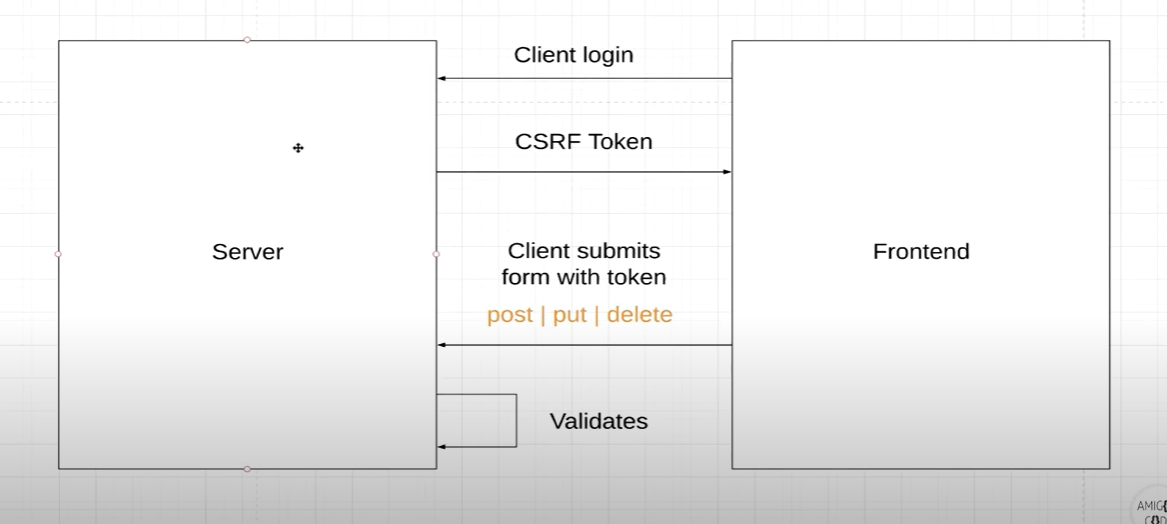
// Using Spring security with Spring Data Jpa   
 ----------------------------------------------   
 @Query("select o from BeerOrder o where o.id=?1 and " +  
 "(true = :#{hasAuthority('order.read')} or " + // : helps hibernate to understand it is a method  
 "o.customer.id = ?#{principal?.customer?.id})")  
 BeerOrder findOrderByIdSecure(UUID orderId);

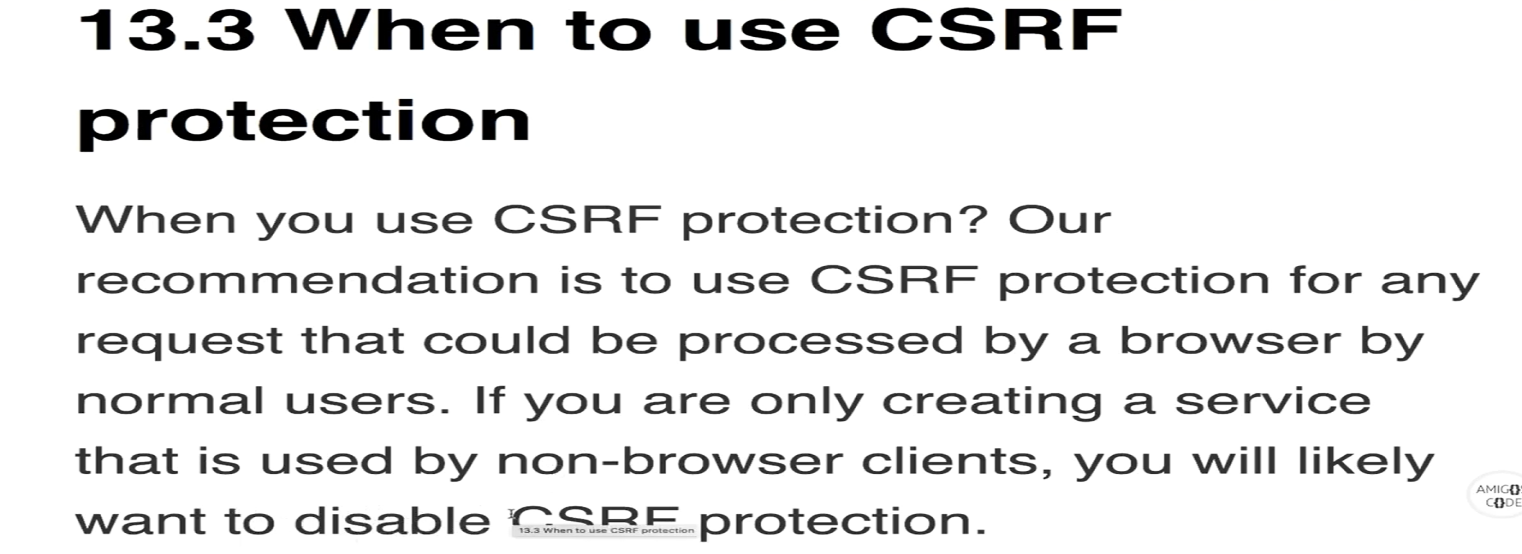
----------------------------------------------  
}

Img:SWJ

8. Enable csrf protection in spring security

YT: <https://www.youtube.com/watch?v=Ub5TLow9GL4> \*\*\* (if you want to understand)



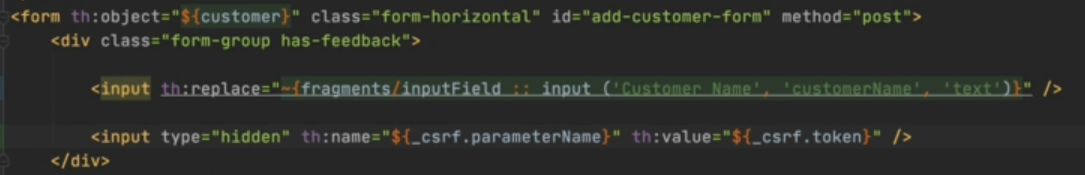


# Ignore csrf() for some endpoints

.csrf().ignoringAntMatchers("/h2-console/\*\*","/api/\*\*");

# Enabling CSRF in html/thymeleaf page.

Adding csrf to a input tag inside a form tag (thymeleaf ex)



In the browser

