## 2. Spring Security and JWT tokens

### Установка Spring Security

В pom.xml нужно добавить зависимость

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

Затем нужно создать конфигурацию и добавить аннотацию @EnableWebSecurity и добавить 2 вспомогательных бина. Первый из них возвращает PasswordEncoder который используется для шифрования паролей

@Bean  
public PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
}  
  
@Bean  
public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {  
 return authenticationConfiguration.getAuthenticationManager();  
}

Затем нужно создать цепочку фильтров, которые буду обрабатывать аутентификацию.

@Bean  
public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {  
 return httpSecurity.build();  
}

1. csrf – cross site request forgery защита отправки аутентифицированных запросов через другие сайты
2. cors – cross origin resource sharing, разрешение доступа к ресурсам сайта на домене отличном от того что использует сайт в данный момент.
3. httpBasics – стандартная авторизация
4. sessionManagement –
5. exceptionHandling – обработка ошибок
   1. authenticationEntryPoint – обработка ошибок аутентификации
   2. accessDeniedHandler – обработка ошибок когда доступ не разрешент
6. authorizeHttpRequests – конфигурирование запросов, какие можно вызвать без авторизации и какие защищены
7. anonymous –
8. addFilter –

### Классы UserDetails и UserDetailsService

**UserDetails** – основная информация об пользователе

public class JwtEntity implements UserDetails {  
  
 private Long id;  
 private final String username;  
 private final String name;  
 private final String password;  
 *// roles and permission of user* private final Collection<? extends GrantedAuthority> authorities;  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

Коллекция **GrantedAuthority** нужна для проверки что пользователь может выполнять и доступ к каким ресурсам у него есть.

**UserDetailsService** – основной интерфейс Spring Security для загрузки данных об пользователе. Содержит единственный метод, который возвращает UserDetails.

public class JwtUserDetailsService implements UserDetailsService {  
 private final UserService userService;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userService.getByUsername(username);  
 return JwtEntityFactory.*create*(user);  
 }  
}

Когда Spring Security будет пытаться авторизовать пользователя, он будет обращаться к методу **loadUserByUsername**, доставать пользователя в виде **JwtEntity** и будет сравнивать пароль с помощью который был захешировать с помощь **PasswordEncoder**

### Получение конфигураций из application.yaml

Для того чтобы облегчить получение данных конфигурации из application.yaml можно использовать зависимость **spring-boot-configuration-processor**.

@Component  
@Data  
@ConfigurationProperties(prefix = "security.jwt")  
public class JwtProperties {  
  
 private String secret;  
 private Long access;  
 private Long refresh;  
}

### Создание и проверка Json Web Token

Для создания и проверки на понадобятся следующие библиотеки: **jjwt-api, jjwt-impl, jjwt-jackson**.

Для начала нужно добавить ключ в properties и считать его в программе в виде байт кода.

// key used to sign tokens  
private SecretKey key;

@PostConstruct  
public void init() {  
 // initializing key in @PostConstruct because we need jwtProperties  
 key = Keys.*hmacShaKeyFor*(jwtProperties.getSecret().getBytes());  
}

Затем нужно использовать **Jwts** для создания и подписания токена. В токе мы передаем id, username, и роли пользователя. Дату когда токен истечет, мы рассчитываем с помощью **Instant**.

// create access (jwt) token and store userId, username and role  
// expires in 1 hour  
public String createAccessToken(Long userId, String username, Set<Role> roles){  
  
 // create claims - data that will be stored in access token  
 Claims claims = Jwts  
 .*claims*()  
 .subject(username)  
 .add("id",userId)  
 .add("roles",resolveRoles(roles))  
 .build();  
  
 // create expiration date time  
 Instant validity = Instant.*now*()  
 .plus(jwtProperties.getAccess(), ChronoUnit.*MILLIS*);  
  
 // create jwt token, set claims, expiration, sign it with key and build  
 return Jwts.*builder*()  
 .claims(claims)  
 .expiration(Date.*from*(validity))  
 .signWith(key)  
 .compact();  
}

Мы также можем прочитать данные из имеющегося токена. Для этого нужно создать парсер, передать в него ключ, которым токен был подписан. Затем мы парсим токен и получаем его содержимое. Из содержимого можно получить поле по названию. Обязательно нужно передать возвращаемый тип, так как по уполчанию **get** возвращает **Object**.

// get any value with access token  
private <T> T getPayloadValue(String token, String name, Class<T> clazz){  
 return Jwts  
 .*parser*()  
 .verifyWith(key)  
 .build()  
 .parseSignedClaims(token)  
 .getPayload()  
 .get(name, clazz);  
}

### Фильтр для Json Web Token

Нужно создать свой Json Web Token Filter, который будет получать Authentication по токену который мы храним в заголовке Authorization.

@AllArgsConstructor  
public class JwtFilter extends GenericFilterBean {  
 private JwtProvider jwtProvider;  
  
 @Override  
 public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  
  
 // read bearer token (access) from Authorization header  
 // we must cast servelRequest to HttpServletRequest to be able to access getHeader method  
 String bearerToken = ((HttpServletRequest)servletRequest).getHeader("Authorization");  
  
 // check if bearer token is not null and it starts with "Bearer "  
 if(bearerToken != null && bearerToken.startsWith("Bearer ")){  
  
 // cut the "Bearer " part from token to get clean access token  
 String accessToken = bearerToken.substring(7);  
  
 // validate access token  
 if(accessToken != null && jwtProvider.validateToken(accessToken)){  
 try{  
  
 // try to get authentication  
 Authentication authentication = jwtProvider.getAuthentication(accessToken);  
  
 // if authentication is not null set it to SecurityContextHolder  
 // it means that user is authenticated  
 if(authentication != null){  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 }  
 }catch (Exception ignored){  
  
 }  
  
 filterChain.doFilter(servletRequest,servletResponse);  
 }  
 }  
 }  
}