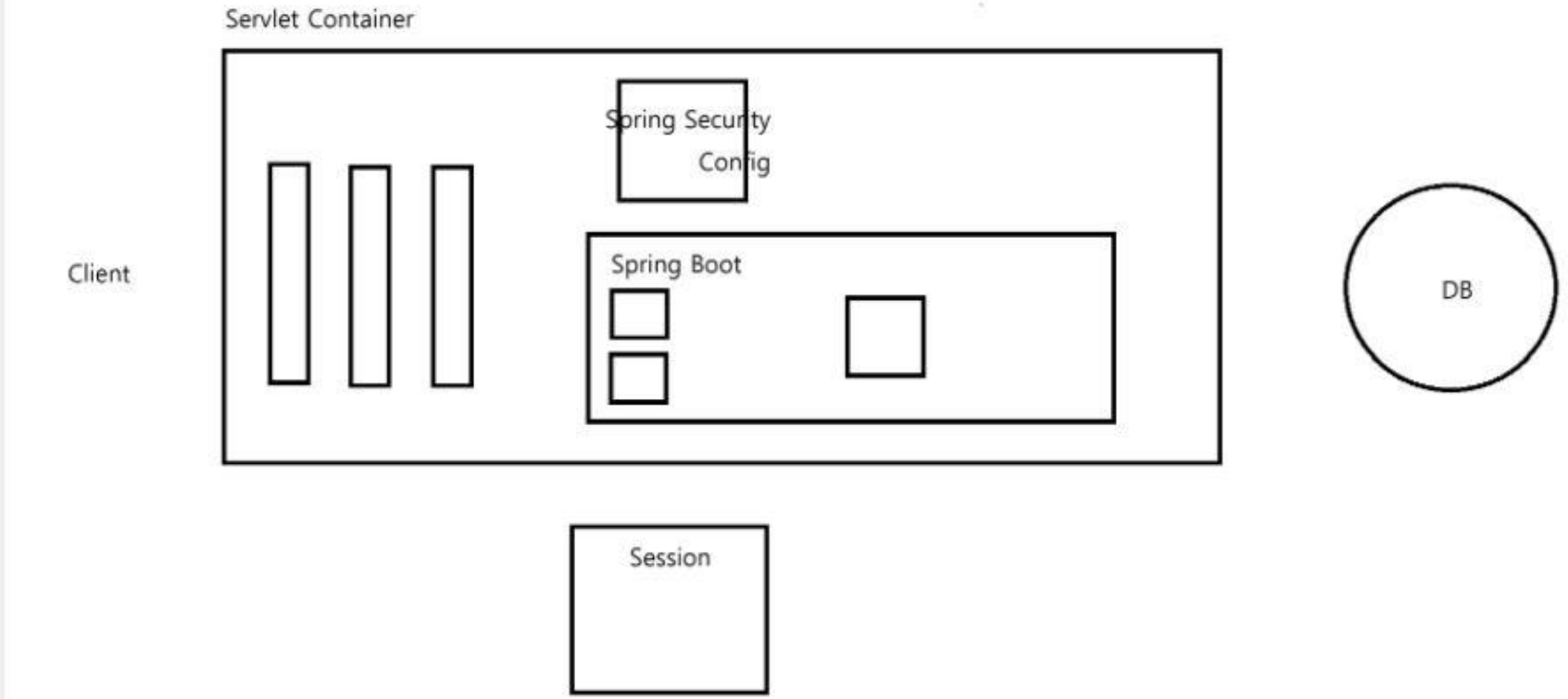
07

Spring Security

01. Security 인증

시큐리티 동작 원리



01. Security 인증

Dependency 추가

```
implementation 'org.springframework.boot:spring-boot-starter-security'
```

인증 방식

- Session 방식
- JWT 토큰 방식

01. Security 인증

Session 방식

- 로그인이 인증되면 유저아이디, 유효기간, session id 등을 서버에 저장
  - session id : 클라이언트를 구분하기 위한 랜덤 문자나 숫자
- SessionID를 클라이언트에게 전달
- 이후 클라이언트가 서버에게 API요청할 때 발급된 SessionID를 쿠키에 담아 서버에 전달
- 서버는 클라이언트가 전달한 SeesionID와 서버에 저장된 SessionID를 비교하여 같으면 요청에 대한 응답을 처리
- 인증정보를 DB에 저장할 경우 속도가 느려질수 있으므로 Redis와 같은 메모리기반 DB를 사용

01. Security 인증

Session 방식 (SecurityConfig.java)

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
…
…
…
…
}
```

01. Security 인증 (Session 방식 (SecurityConfig.java))

```
@Bean
    public SecurityFilterChain configure(HttpSecurity http) throws Exception {
        http.csrf(csrf->csrf.disable())
                .authorizeHttpRequests(authorizeRequests ->{
                    authorizeRequests.requestMatchers("/","/login", "/join").permitAll();
                    authorizeRequests.requestMatchers("/admin").hasRole("ADMIN");
                    authorizeRequests.requestMatchers("/user").hasAnyRole("USER", "ADMIN");
                    authorizeRequests.anyRequest().authenticated();
                })

                .formLogin(formLogin ->
                        formLogin.loginProcessingUrl("/login")
                        .successHandler(authenticationSuccessHandler())
                        .failureHandler(authenticationFailureHandler())
                )

                .cors(cors->cors.configurationSource(request -> {
                    CorsConfiguration corsConfiguration = new CorsConfiguration();
                    corsConfiguration.addAllowedOrigin("http://localhost:3000");
                    corsConfiguration.addAllowedHeader("*");
                    corsConfiguration.addAlloeredMethod("*"));
                    corsConfiguration.setAllowCredentials(true);
                    return corsConfiguration;
                        }));
        return http.build();
```

01. Security 인증 (Session 방식)

SecurityConfig.java

```
@Bean
public AuthenticationSuccessHandler authenticationSuccessHandler() {
    return((request, response, authentication) -> {
        Map<String, Object> responseData = new HashMap<>();
        responseData.put("result", "로그인  성공");

        ObjectMapper objectMapper = new ObjectMapper();
        String jsonMessage = objectMapper.writeValueAsString(responseData);

        response.setStatus(200);
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().write(jsonMessage);
    });
}
```

01. Security 인증 (Session 방식)

SecurityConfig.java

```
@Bean
public AuthenticationFailureHandler authenticationFailureHandler() {
    return((request, response, exception) -> {
        Map<String, Object> responseData = new HashMap<>();
        responseData.put("result", "로그인  실패");

        ObjectMapper objectMapper = new ObjectMapper();
        String jsonmessage = objectMapper.writeValueAsString(responseData);

        response.setStatus(401); // HTTP 401 Unauthorized
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().write(jsonmessage);
    });
}
```

01. Security 인증 (Session 방식)

SecurityConfig.java

```
@Bean
public LogoutSuccessHandler logoutSuccessHandler() {
    return((request, response, authentication) -> {
        response.setStatus(200);
        response.getWriter().write("Logout success");
    });
}
```

```
.logout(logout ->
        logout.logoutUrl("/logout")
        .logoutSuccessHandler(logoutSuccessHandler())
        .addLogoutHandler((request, response, authentication) -> {
            if(request.getSession()!=null){
                request.getSession().invalidate(); // 세션정보 무효화
            }
            SecurityContextHolder.clearContext();
                        //저장된 인증 정보 삭제(ThreadLocal에 따로 저장된 정보)
            })
        .deleteCookies("JSESSIONID")
        )
```

01. Security 인증 (Session 방식)

SecurityConfig.java

```
.sessionManagement(auth->
        auth.maximumSessions(1)
                .maxSessionsPreventsLogin(false)
                        //중복 로그인 허용하되, 기존 세션을 만료시킴 ..
                        //(true): 나중에 접속하는 세션을 막는다
               .expiredSessionStrategy(event -> {//만료된 세션과 관련된 정보
                   HttpServletResponse response = event.getResponse();
                    //만료된 세션과 관련된 HTTP 응답 객체
                   response.setContentType("application/json");
                     response.setCharacterEncoding("UTF-8");
                     SecurityContextHolder.clearContext();
                     response.getWriter().write("다른 호스트에서 로그인하여 현재
                   세션이 만료되었습니다.");
                 }));
```

01. Security 인증 (Session 방식)

CSRF TOKEN 활성화

-AuthenticationSuccessHandler()

```
CsrfToken token = (CsrfToken) request.getAttribute(CsrfToken.class.getName());
responseData.put("csrf-token", token.getToken());
```

-AuthenController

```
@GetMapping(value = "/csrf-token")
public ResponseEntity<Map<String, String>> csrf(HttpServletRequest request) {
    CsrfToken token = (CsrfToken)
                        request.getAttribute(CsrfToken.class.getName());
    Map<String, String> map = new HashMap<>();
    map.put("csrf-token", token.getToken());
    System.out.println(token.getToken());
    return ResponseEntity.ok(map);
}
```

01. Security 인증 (Session 방식)

UserAuthenticationService.java

```
@Service
@RequiredArgsConstructor
public class UserAuthenticationService implements UserDetailsService {
    private final AuthRepository authRepository;
    @Override
    public UserDetails loadUserByUsername(String username) throws
            UsernameNotFoundException {
        AuthEntity authEntity = this.authRepository.findByUsername(username);
        if(authEntity == null) {
            throw new UsernameNotFoundException("User not found" + username);
        }
        List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
        grantedAuthorities.add(new SimpleGrantedAuthority(authEntity.getRole()));
        return new User(authEntity.getUsername(), authEntity.getPassword(),
                        grantedAuthorities);
    }
}
```

01. Security 인증 (Session 방식)

CSRF TOKEN 활성화 – frontend

```
const response=await axios.get("http://localhost:8080/admin",
    {
            withCredentials:true, // 세션정보 쿠키포함
            headers:{
        'X-CSRF-TOKEN':csrfToken, // 헤더에 Csrf 토큰 추가
    }});
```