Семинар №7
Название семинара

1. **Инструментарий:**

Урок
Презентация

---

2. **Цели семинара №7:**

- Получить базовые знания по использованию Spring Security.
- Закрепить понимание принципов безопасности и защиты от атак при работе с JWT.

По итогам семинара №7 слушатель должен **знать**:

- Основные принципы работы Spring Security.
- Как интегрировать JWT в Spring Security.

По итогам семинара №7 слушатель должен **уметь**:

- Настроить Spring Security.
- Реализовать систему аутентификации и авторизации на основе JWT.
- Защищать свои приложения от основных видов атак, связанных с JWT

---

3. **План Содержание:**

Этап урока	Тайминг, минуты	Формат
Введение, обзор темы	20	Модерирует преподаватель
Задание 1	40	Студенты выполняют, преподаватель помогает в решении проблем
Задание 2	40	Студенты выполняют,

		преподаватель помогает в решении проблем
Вопросы и обсуждение	20	Модерирует преподаватель
Длительность:	**120**	

4. Блок 1.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 30 минут

Задание:

Вам необходимо реализовать базовую аутентификацию и авторизацию в вашем Spring Boot приложении с использованием Spring Security. Ваше приложение должно содержать две роли: USER и ADMIN.

1. Любой аутентифицированный пользователь (роль USER) должен иметь доступ к эндпоинту /user-info, который возвращает информацию о текущем пользователе.
2. Только пользователи с ролью ADMIN должны иметь доступ к эндпоинту /admin-dashboard, который возвращает простое административное сообщение.

Пример решения:

1. Конфигурация безопасности:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService)
            .passwordEncoder(new BCryptPasswordEncoder());
    }

    @Override
```

```
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/user-info").hasRole("USER")
            .antMatchers("/admin-dashboard").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and()
            .httpBasic();
    }
}
```

2. Контроллер:

```
@RestController
public class AppController {

    @GetMapping("/user-info")
    public ResponseEntity<String> userInfo(Authentication authentication) {
        return ResponseEntity.ok("Welcome, " + authentication.getName() + "!");
    }

    @GetMapping("/admin-dashboard")
    public ResponseEntity<String> adminDashboard() {
        return ResponseEntity.ok("This is the admin dashboard.");
    }
}
```

Часто встречающиеся ошибки:

1. Неправильная конфигурация ролей: Часто новички забывают добавлять префикс "ROLE_" при конфигурации прав доступа, что приводит к ошибке доступа.
2. Отсутствие шифрования паролей: Не использовать PasswordEncoder или использовать простое хранение паролей в открытом виде. Это серьезная угроза безопасности.
3. Отключение всех механизмов безопасности: Например, полное отключение CSRF, что может сделать приложение уязвимым для атак.
4. Отсутствие обработки исключений: Если пользователь пытается получить доступ к ресурсу, к которому у него нет доступа, приложение должно корректно обрабатывать такие ситуации, например, возвращать соответствующий код ошибки.

5. Блок 2.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 20 минут

Задание:

Реализуйте регистрацию и вход в вашем Spring Boot приложении с использованием Spring Security и JWT (JSON Web Tokens). У вас должны быть следующие эндпоинты:

1. /register - эндпоинт для регистрации новых пользователей. Принимает имя пользователя и пароль.
2. /login - эндпоинт для входа. Принимает имя пользователя и пароль, и если они корректны, возвращает JWT.
3. /dashboard - защищенный эндпоинт, который доступен только для аутентифицированных пользователей с валидным JWT. Возвращает сообщение "Welcome to the dashboard!"

Пример решения:

1. Конфигурация безопасности:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private UserDetailsService userDetailsService;

  @Autowired
  private JwtRequestFilter jwtRequestFilter;

  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService)
      .passwordEncoder(new BCryptPasswordEncoder());
  }

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
```

```
            .csrf().disable()
            .authorizeRequests()
            .antMatchers("/login", "/register").permitAll()
            .anyRequest().authenticated()
            .and().sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
    }
}
```

2. Контроллер:

```
@RestController
@RequestMapping("/api")
public class AppController {

    @Autowired
    private AuthenticationService authService;

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody User user) {
        authService.register(user);
        return ResponseEntity.ok("User registered successfully!");
    }

    @PostMapping("/login")
    public ResponseEntity<?> createAuthenticationToken(@RequestBody
AuthenticationRequest authenticationRequest) throws Exception {
        final String jwt = authService.login(authenticationRequest);
        return ResponseEntity.ok(new JwtResponse(jwt));
    }

    @GetMapping("/dashboard")
    public ResponseEntity<String> dashboard() {
        return ResponseEntity.ok("Welcome to the dashboard!");
    }
}
```

Часто встречающиеся ошибки:
1. Неуправляемые JWT: Часто разработчики не реализуют механизмы для отзыва или обновления токенов, что может создать проблемы с безопасностью, особенно если токен утрачен или скомпрометирован.
2. Хранение секретов в коде: Хранение секрета для подписи JWT прямо в исходном коде, что делает систему уязвимой.
3. Отсутствие обработки исключений: Если в процессе аутентификации или валидации JWT происходит ошибка, приложение должно корректно обрабатывать такие ситуации и сообщать пользователю об ошибке.
4. Пропуск валидации входящих данных: Недостаточная проверка и валидация входящих данных при регистрации или входе может привести к уязвимостям.

6. Домашнее задание

Условие:

Вам необходимо создать Spring Boot приложение, которое управляет доступом к ресурсам в зависимости от роли пользователя. У вас должно быть два типа пользователей: USER и ADMIN.

1. Создайте ресурс /private-data доступный только для аутентифицированных пользователей с ролью ADMIN.
2. Создайте ресурс /public-data доступный для всех аутентифицированных пользователей независимо от их роли.
3. Реализуйте форму входа для аутентификации пользователей с использованием стандартных средств Spring Security.
4. Если неаутентифицированный пользователь пытается получить доступ к /private-data, он должен быть перенаправлен на форму входа.

Пример решения:

1. Конфигурация безопасности:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private UserDetailsService userDetailsService;

  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService)
      .passwordEncoder(new BCryptPasswordEncoder());
```

```
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/public-data").hasAnyRole("USER", "ADMIN")
            .antMatchers("/private-data").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and()
            .formLogin().permitAll();
    }
}
```

2. Контроллер:

```
@RestController
public class DataController {

    @GetMapping("/public-data")
    public String publicData() {
        return "This is public data.";
    }

    @GetMapping("/private-data")
    public String privateData() {
        return "This is private data. Only for admins!";
    }
}
```

Рекомендации для преподавателей по оценке задания:

1. Структура проекта: Проверьте, правильно ли организован проект и разделены ли слои приложения.
2. Качество кода: Убедитесь, что код чистый, соблюдены отступы, и нет лишних комментариев.
3. Конфигурация безопасности: Проверьте, правильно ли настроены права доступа к ресурсам и использованы ли соответствующие методы конфигурации.
4. Обработка ошибок: Убедитесь, что при попытке неаутентифицированного пользователя получить доступ к /private-data происходит перенаправление на форму входа.
5. Тестирование: Оцените, есть ли юнит-тесты и/или интеграционные тесты, и насколько они покрывают функциональность приложения.