УДК 004

Авторизация с помощью Spring Security

Семченко Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассмотрена возможность защиты приложений с помощью авторизации, доступной в Spring Security. Описан процесс создания защиты и окна авторизации.
Ключевые слова: Spring Security, Java, авторизация

Spring Security authorization

Semchenko Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses the ability to protect applications using authorization available in Spring Security. The process of creating protection and authorization window is described.
Keywords: Spring Security, Java, Authorization

Безопасность - важный нефункциональный аспект приложения. Безопасное приложение обычно аутентифицирует пользователей, а также может проверять авторизацию для выполнения задач в зависимости от роли пользователя. Аутентификация и авторизация - это основные компоненты, которые определяют, может ли пользователь получить доступ и выполнить задачу в приложении.

Цель данной статьи рассмотреть возможности Spring Security и создать окно авторизации.

С.В. Мельников провел обзор на работу с отладочным интерфейсом Java и методом модификации функциональности приложения, не изменяющий его бинарные файлы [1]. А.А.Шейн, Д.Г.Залевский, С.В.

Автайкин, С.В.Карташев, С.А.Скороход разработали и описали действия программы предназначенной для автоматического создания набора классов для представления объектов модели Decode в виде нативных объектов языка Java [2]. Н.Н. Глибовец проанализировала в своей работе особенности агентных технологий и перспективы их использования для разработки сложных многопользовательских программных систем [3]. В своей работе М.К.Ермаков, С.П.Вартанов рассмотрели вопросы проведения анализа программ интерпретируемых языков программирования [4]. Так же А.А. Птицын, Н.Л. Подколодный, Д.А.Григорович, С.В. Лаврюшев разработали молекулярно-биологический сервер и на его базе создали ряд информационно-вычислительных систем, для изучения регуляции экспрессии генов с использованием новейших технологий Java [5].

Платформа безопасности Spring Security обеспечивает полную поддержку функций аутентификации и авторизации приложений.

Она состоит из таких компонентов как:

- Фильтр аутентификации: фильтр аутентификации принимает запросы пользователей и пересылает их диспетчеру аутентификации для аутентификации.
- Диспетчер аутентификации: диспетчер аутентификации использует поставщика аутентификации для выполнения процесса аутентификации.
- Провайдер аутентификации: провайдер аутентификации проверяет, существует ли пользователь в системе с помощью службы сведений о пользователе, и проверяет пароль с помощью кодировщика паролей.
- Сервис сведений о пользователях: реализует ответственность за управление пользователями. Он ищет существующего пользователя, который используется провайдером для аутентификации пользователя.
- Кодировщик паролей: кодирует пароль перед сохранением и сопоставляет закодированный пароль с типом кодирования.
- Контекст безопасности: контекст безопасности содержит сведения о пользователе после успешного процесса аутентификации и авторизации.

Безопасность Spring - это гибко настраиваемая среда. В ней можно настроить любую часть конфигурации фреймворка, например логику аутентификации, индивидуальное представление сведений о пользователе и т.д.

Теперь попробуем создать простой пример реализации авторизации. Создадим загрузочное приложение Spring с указанными зависимостями (рис.1).

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Рисунок 1 – pom.xml

Далее создадим REST-контроллер, он будет отображать стартовое окно после успешной авторизации (рис.2).

```
import ...

@RestController
public class HelloController {

    @GetMapping("/")
    public String getMessage() { return ""; }

}
```

Рисунок 2 - Контроллер

Теперь запустив приложение получаем окно авторизации, где необходимо ввести логин и пароль (рис.3).

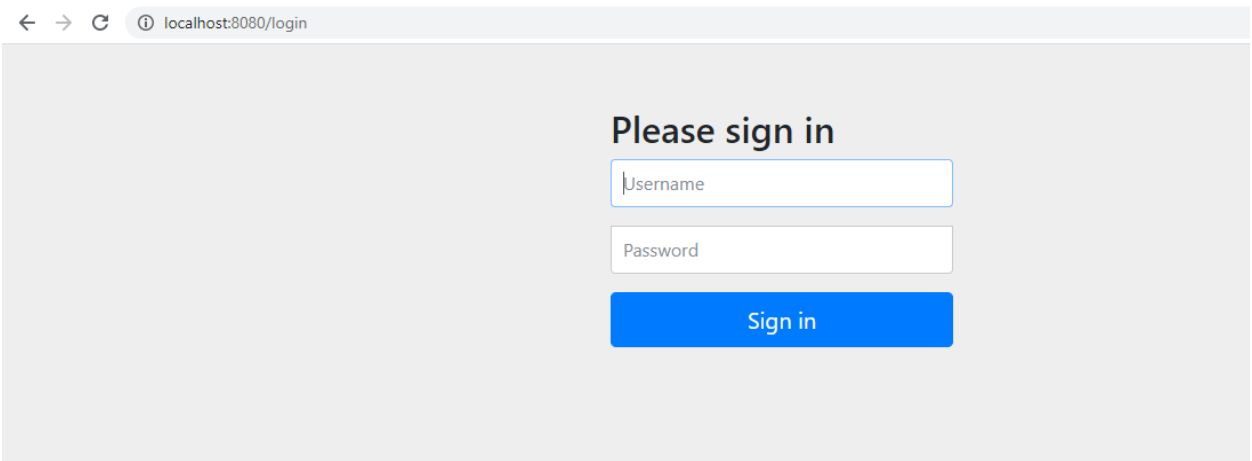

Рисунок 3 – Окно авторизации

Логин по умолчанию становится Admin, а пароль случайным образом генерируется во время запуска приложения и выводится в консоль (рис.5).

```
2020-12-05 22:13:01.467  INFO 3100 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecu
2020-12-05 22:13:02.326  INFO 3100 --- [           main] .s.s.UserDetailsServiceAutoConfigura

Using generated security password: 70d74625-7373-4975-b171-4434735b66da

2020-12-05 22:13:02.964  INFO 3100 --- [           main] o.s.s.web.DefaultSecurityFilterChain
2020-12-05 22:13:03.328  INFO 3100 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebSer
2020-12-05 22:13:03.374  INFO 3100 --- [           main] c.a.SpringSecurityExampleApplicatior
```

Рисунок 4 – Вывод пароля

Настройка безопасности Spring проста и аналогична другим фреймворкам Spring. Так же можно определять пользовательские конфигурации или переопределять реализацию по умолчанию, чтобы настроить определенные части платформы в соответствии с требованиями приложения.

Можно переопределить имя пользователя и пароль, добавив следующие свойства в файл конфигурации «application.properties» загрузочного приложения Spring в каталоге / src / main / resources / (рис.5).

```
1  spring.security.user.name=pavel
2  spring.security.user.password=test
```

Рисунок 5 - application.properties

Теперь попробуем использовать реализацию «InMemoryUserDetailsManager» для создания нового пользователя (рис.6).

```
@Configuration
public class CustomSecurityConfig extends WebSecurityConfigurerAdapter {


    @Bean
    public UserDetailsService userDetailsService() {
        var userDetailsService = new InMemoryUserDetailsManager();
        var user : UserDetails = User.withUsername("pavel").password("12345").authorities("read").build();
        userDetailsService.createUser(user);
        return userDetailsService;
    }

    @Bean
    public PasswordEncoder passwordEncoder() { return NoOpPasswordEncoder.getInstance(); }
```

Рисунок 6 - InMemoryUserDetailsManager

Здесь был создан новый пользовательский экземпляр с именем пользователя «pavel», паролем «12345» и ролью «read». Использован экземпляр «InMemoryUserDetailsManager» (который реализует интерфейс «UserDetailsService») для создания нового экземпляра пользователя.

Также необходимо определить bean-компонент «PasswordEncoder», если определять настраиваемую службу сведений о пользователях. И

определили bean-компонент «NoOpPasswordEncoder», который не применяет кодировку к паролю пользователя.

Чтобы переопределить конфигурацию безопасности, можно создать собственный класс конфигурации безопасности и переопределить требуемый метод «configure()» класса «WebSecurityConfigurerAdapter» (рис.7).

```
@Configuration
public class CustomSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.formLogin().defaultSuccessUrl( defaultSuccessUrl: "/home",  alwaysUse: true);
        http.authorizeRequests().anyRequest().authenticated();
    }
```

Рисунок 7 - WebSecurityConfigurerAdapter

В приведенном выше примере был установлен URL-адрес на «/ home». Приведенная конфигурация применяет безопасность приложения ко всем входящим запросам.

Если перезапустить приложение, то будет перенаправление на URL-путь «/home» после успешного процесса аутентификации, и отобразится экран ошибки по умолчанию, поскольку для этого URL-пути нет обработчика (рис.8).

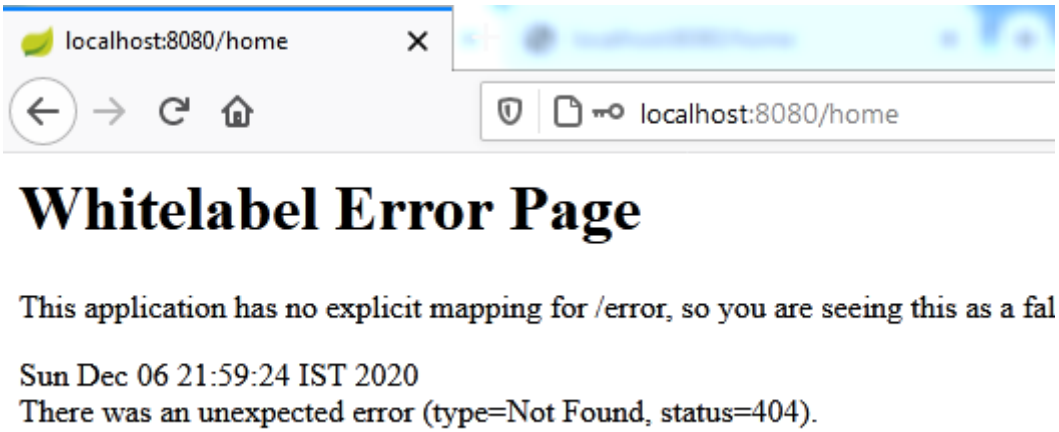

Рисунок 8 – Ошибка пути

Чтобы настроить логику аутентификации, необходимо выполнить следующие шаги:

- Создать собственный класс и реализовать интерфейс «AuthenticationProvider».
- Реализовать метод «authenicate()» и «support()».
- Внедрить настраиваемый поставщик в класс конфигурации безопасности и настроить приложение для использования

внедренного экземпляра настраиваемого поставщика с помощью метода «configure()».

Создадим настраиваемый класс поставщика аутентификации под названием «CustomAuthenticationProvider». Отметим класс аннотацией «@Component», чтобы он мог внедряться в класс конфигурации безопасности (рис.9).

```
@Component
public class CustomAuthenticationProvider implements AuthenticationProvider {

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {

        String username = authentication.getName();
        String password = String.valueOf(authentication.getCredentials());

        if ("pavel".equals(username) && "12345".equals(password)) {
            return new UsernamePasswordAuthenticationToken(username, password, Arrays.asList());
        } else {
            throw new BadCredentialsException("authentication Failed!!");
        }
    }

    @Override
    public boolean supports(Class<?> authentication) {
        return UsernamePasswordAuthenticationToken.class.isAssignableFrom(authentication);
    }
}
```

Рисунок 9 - CustomAuthenticationProvider

Необходимо переопределить метод «Authenticate()», который обрабатывает всю логику, связанную с аутентификацией. Если аутентификация прошла успешно, то возвращаем реализацию объекта аутентификации, который содержит все детали аутентификации. Безопасность Spring регистрирует эти детали аутентификации в контексте безопасности. В приведенном выше примере был возвращен экземпляр аутентификации типа «UsernamePasswordAuthenticationToken», который представляет объект аутентификации вошедшего в систему пользователя. Метод «supports()» проверяет поддерживаемый тип аутентификации.

В примере идет проверка, есть ли у объекта «Authentication» имя пользователя и пароль, и он представляет класс реализации аутентификации «UsernamePasswordAuthenticationToken» .

Если учетные данные неверны, приложение выдает исключение «BadCredentialsException» с сообщением (рис.10).

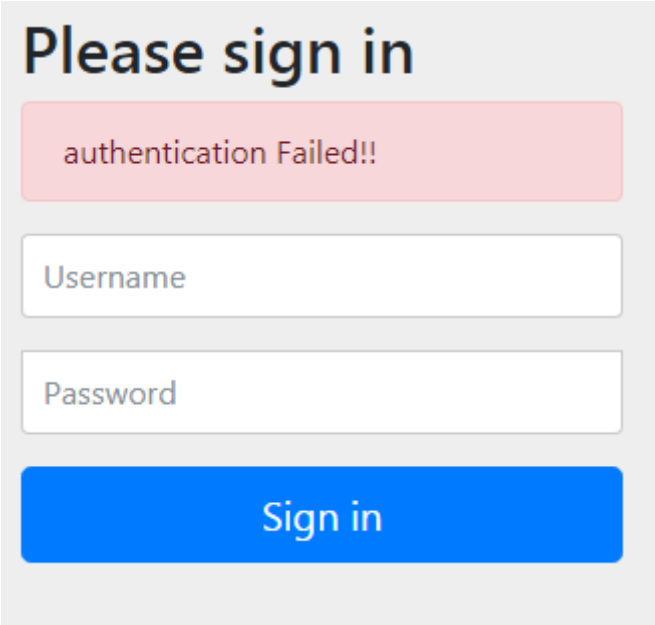

Рисунок 10 - BadCredentialsException

Настроим класс конфигурации безопасности, внедрив поставщика проверки подлинности (рис.11).

```
@Configuration
public class CustomSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private CustomAuthenticationProvider authProvider;

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.formLogin().defaultSuccessUrl( defaultSuccessUrl: "/home",  alwaysUse: true);
        http.authorizeRequests().anyRequest().authenticated();
    }

    @Override
    public void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(authProvider);
    }

    @Bean
    public UserDetailsService userDetailsService() {
        var userDetailsService = new InMemoryUserDetailsManager();
        var user : UserDetails  = User.withUsername("pavel").password("12345").authorities("read").build();
        userDetailsService.createUser(user);
        return userDetailsService;
    }

    @Bean
    public PasswordEncoder passwordEncoder() { return NoOpPasswordEncoder.getInstance(); }
}
```

Рисунок 11 - CustomAuthenticationProvider

В этой статье были рассмотрены основы безопасности Spring, как внутренняя безопасность Spring обрабатывает аутентификацию и как настроить некоторые параметры безопасности Spring.

Библиографический список

1. Шейн А.А., Залевский Д.Г., Автайкин С.В., Карташев С.В., Скороход С.А. Генератор исходного кода на языке java по описанию бортовых компонентов decode (decode java generator 0.2) // Вестник волжского университета им. в.н. татищева. 2019. №3. С. 26-32.
2. Мельников С.В. Обзор и применение отладочного интерфейса java (jdi) для обратимой модификации программных продуктов // Современные проблемы науки и образования. 2018. №8. С. 8-19.
3. Глибовец Н.Н. Использование jade (java agent development environment) для разработки компьютерных систем поддержки дистанционного обучения агентного типа // Заметки по информатике и математике. 2019. №10. С. 15-20.
4. Ермаков М.К., Вартанов С.П. Подход к проведению динамического анализа java-программ методом модификации виртуальной машины java // Научные труды Винницкого национального технологического университета. 2018. №6. С. 10-17.
5. Птицын А.А., Подколодный Н.Л., Григорович Д.А., Лаврюшев С.В. Создание молекулярно-биологического сервера www с использованием новейших технологий java // Заметки по информатике и математике. 2020. №1. С. 11-20.