УДК 004

Аутентификация JWT на основе ролей в SpringBoot

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассмотрен и описан процесс разработки простой программы с авторизацией JWT на основе ролей. Разработка происходит на языке программирования Java. Итоговым результатом является созданная схема программы, выполняющая защищенность вэб сервиса.
Ключевые слова: Аутентификация, Java, Spring

JWT authentication based on roles in Springboot

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses and describes the process of developing a simple program with JWT's authorization based on roles. Development takes place in the Java programming language. The final result is the created program scheme that performs the security of the VEB service.
Keywords: Authentication, Java, Spring

Настройка аутентификации и авторизации всегда была самой запутанной задачей в java. Она включает в себя множество действий, и любой разработчик будет постоянно путаться, пока не повторит эту процедуру множество раз.

Цель данной статьи – разработать схему аутентификации JWT на основе ролей.

А.В. Колойдчук рассмотрела методы программирования кредитных рисков внедрения ИКТ с помощью Java, Pascal и Basic [1]. А.А. Симаков

описал разработку трассировки для JVM программ в использовании анализа ПО и обратного проектирования [2]. А.Н. Иванов описал процесс реализации безопасного веб-приложения на Java [3]. В своей работе В.П. Великов, К.С. Добрева рассмотрели проблемы автоматизированной генерации ПО [4]. В своей статье А.А. Шейн описал процесс разработки программы, которая автоматически создает наборы классов в виде нативных объектов Java [5].

Различные способы аутентификации включают в себя: базовую аутентификацию по имени пользователя и паролю, аутентификацию на основе токенов, ключ API, O-Auth.

Создадим простое приложение Springboot с конечными точками для доступа или обновления данных университета. Реализуем чтобы доступ к этим конечным точкам осуществлялся только с правильным токеном. Для JWT будет отдельная таблица для хранения учетных данных и ролей.

Прежде всего, нужны библиотеки безопасности Spring. Включим приведенную ниже зависимость в файл pom.xml (рис.1).

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

Рисунок 1 – Библиотека безопасности

В этой библиотеке есть один абстрактный класс «WebSecurityConfigurerAdapter», с помощью которого можно настроить безопасность всего приложения. По умолчанию аутентификация отключена. Чтобы включить его, нужно настроить реализацию методов в этом классе.

Расширим класс «WebSecurityConfigurerAdapter» и переопределим методы:

- «authenticationManagerBean()» - для получения Bean-компонента «AuthenticationManager»;
- «configure» - чтобы разрешить или ограничить любую конечную точку для аутентификации.

```java
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

    @Autowired
    private UserDetailsService jwtUserDetailsService;

    @Autowired
    private JwtRequestFilter jwtRequestFilter;

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

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.headers().frameOptions().disable();
        httpSecurity.authorizeRequests().antMatchers( ...antPatterns: "/h2-console/*").permitAll();
        httpSecurity.csrf().disable()
                .authorizeRequests().antMatchers( ...antPatterns: "/authenticate").permitAll().
                        anyRequest().authenticated().and()
                .exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);
    }

```

Рисунок 2 – WebSecurityConfig

В приведенном выше рисунке аннотация «@EnableWebSecurity» используется для включения безопасности приложения. Поскольку используется база данных H2, необходимо разрешить доступ к ней.

Для каждого запроса к приложению проверка токена и настройка аутентификации в контексте будут выполняться в настроенном фильтре «JwtRequestFilter.java», добавленном перед «UsernamePasswordAuthenticationFilter».

Настроим таблицы базы данных H2 в памяти с помощью сценариев SQL. В файле «application.properties» укажим сведения о подключении к базе данных H2, токен JWT и номер порта (рис.3).

```
server.port=8082

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.settings.web-allow-others=true
spring.jpa.hibernate.ddl-auto=update

spring.messages.basename=messages

management.endpoints.web.exposure.include=*

jwt.secret=mytoken
```

Рисунок 3 – application.properties

Реализуем загрузку таблиц после запуска приложения. Как упоминалось выше, у приложения будет две таблицы, создадим их (рис.4) и заполним некоторыми данными (рис.5).

```
CREATE TABLE UserData (
    id INT AUTO_INCREMENT  PRIMARY KEY,
    username VARCHAR(250) NOT NULL,
    password VARCHAR(250) NOT NULL,
    role VARCHAR(250) NOT NULL
);

CREATE TABLE University (
    id INT AUTO_INCREMENT  PRIMARY KEY,
    uname VARCHAR(250) NOT NULL,
    address VARCHAR(250) NOT NULL
);
```

Рисунок 4 – schema.sql

```
INSERT INTO UserData (id, username, password,role) VALUES
    (1,'Pavel','Pavel','ROLE_USER,ROLE_ADMIN'),
    (2,'Andrey', 'Andrey','ROLE_ADMIN'),
    (3,'Oleg', 'Oleg','ROLE_ADMIN'),
    (4,'Evgeniy', 'Evgeniy','ROLE_EDITOR'),
    (5,'Olga', 'Olga','ROLE_PUBLIC');

INSERT INTO University (id, uname, address) VALUES
    (1,'NSU','Novosibirsk'),
    (2,'MSU','Moscow'),
    (3,'PSU','Birobidzhan')
```

Рисунок 5 – data.sql

Кроме того, создадим соответствующие классы POJO: UserData и UniversityDB и интерфейсы UserDataRepo и UniversityRepo, реализующие JpaRepository для выполнения операций DAO.

Пусть будет одна конечная точка POST («/authenticate») для целей аутентификации, которая принимает имя пользователя и пароль. Если они подлинные, он возвращает токен. Возвращенный токен можно использовать для доступа к конечным точкам приложения до истечения срока действия.

Учетные данные, переданные в конечной точке аутентификации, будут проверены в компоненте UserDetailsService (рис.6).

```java
@Service
public class JwtUserDetailsService implements UserDetailsService {

    @Autowired
    private UserDataRepo userDataRepo;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserData userData = userDataRepo.findByUsername(username);
        System.out.println("userData:"+ userData);
        if (userData != null) {
            Collection<String> mappedAuthorities = Arrays.asList(userData.getRole().split( regex: ","));
            User user = new User(username, new BCryptPasswordEncoder().encode(userData.getPassword()), mappedAuthorities.stream().map(SimpleGrantedAuthority::new).collect(Collectors.toList()));
            return user;
        } else {
            throw new UsernameNotFoundException("User not found with username: " + username);
        }
    }
}
```

Рисунок 6 - JwtUserDetailsService

Создадим конечную точку «/authenticate», которая принимает учетные данные для входа в тело, подтверждая имя пользователя и пароль. Настроим на получение всех сведений о пользователе из базы данных H2 и генерацию токен, используя jwt-secret из файла свойств. В качестве ответа будет отправлен токен (рис.7).

```java
@RestController
public class JwtAuthenticationController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private JwtUserDetailsService userDetailsService;

    @RequestMapping(value = "/authenticate", method = RequestMethod.POST)
    public ResponseEntity<?> createAuthenticationToken(@RequestBody UserData authenticationRequest) throws Exception {

        authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

        final UserDetails userDetails = userDetailsService
                .loadUserByUsername(authenticationRequest.getUsername());

        final String token = jwtTokenUtil.generateToken(userDetails);

        return ResponseEntity.ok(new JwtResponse(token));
    }

    private void authenticate(String username, String password) throws Exception {
        try {
            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));
        } catch (DisabledException e) {
            throw new Exception("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new Exception("INVALID_CREDENTIALS", e);
        }
    }
}
```

Рисунок 7 – JwtAuthenticationController

Теперь создадим сами конечные точки, благодаря которым сможем получить ответ после предоставления токена в вызовах API (рис.8).

```java
@RestController
@RequestMapping("/university")
public class UniversityController {

    @Autowired
    private UniversityRepo universityRepo;

//    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/test")
    public String test() { return "in university test endpoint"; }

//    @PreAuthorize("hasAnyRole('ADMIN','EDITOR')")
    @PostMapping
    public UniversityDB addUniversity(@RequestBody UniversityDB universityDB) {
        UniversityDB savedUniversity = universityRepo.save(universityDB);
        return savedUniversity;
    }

//    @PreAuthorize("hasAnyRole('ADMIN','EDITOR','USER','PUBLIC')")
    @GetMapping
    public List<UniversityDB> getUniversities() {
        List<UniversityDB> allUniversitydata = universityRepo.findAll();
        return allUniversitydata;
    }
}
```

Рисунок 8 – UniversityController

Теперь, когда пользователь прошел аутентификацию для доступа к приложению. В зависимости от своей роли он получит доступ к частям приложения. Например, только администратор может получить доступ к конечным точкам DELETE, пользователи с ролью EDITOR могут вызывать только конечные точки POST, а другие обычные пользователи могут только просматривать данные, вызывать только конечные точки GET.

Теперь только пользователь с указанной ролью может получить доступ к конечным точкам приложения. При попытке подключится без данных аутентификации будет выходить ошибка 401 (рис.9).

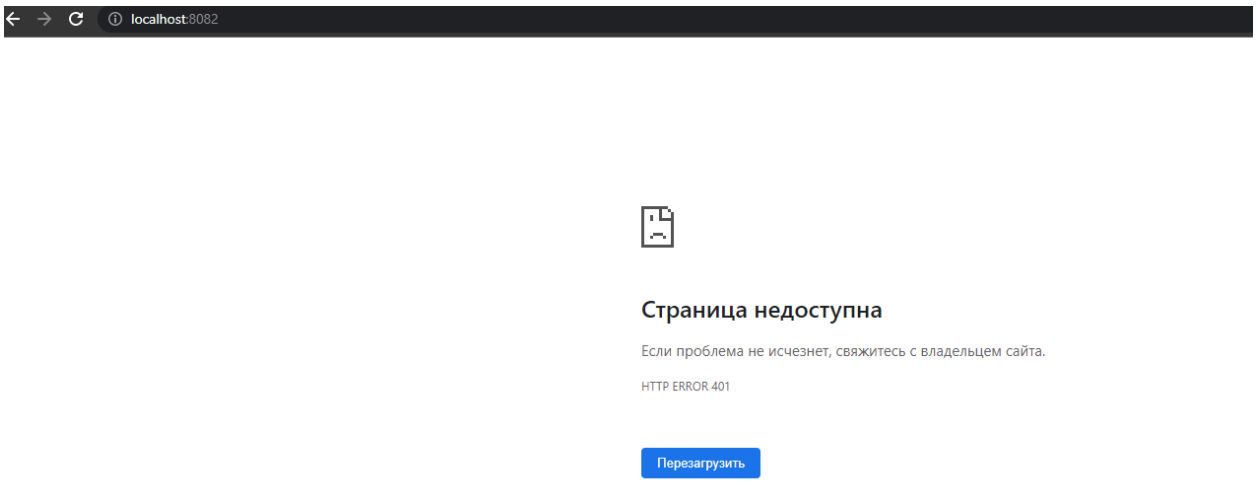

Рисунок 9 – Запрет доступа

И наоборот, при авторизации с использованием логина и пароля, будет подсказка о разрешённом доступе (рис.10).

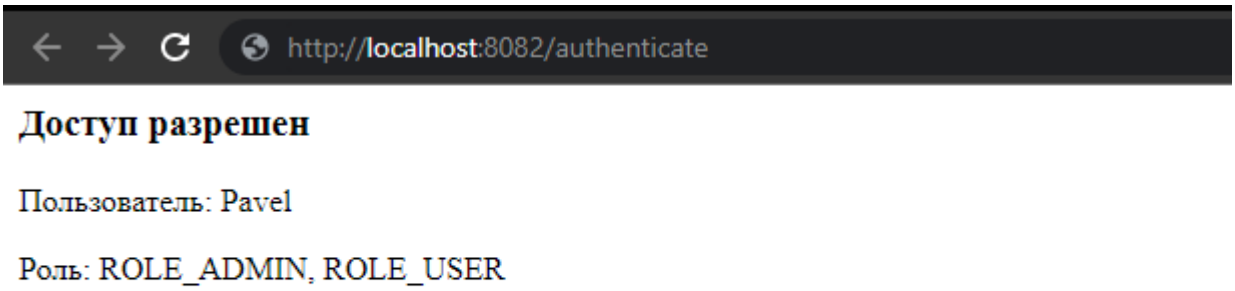

Рисунок 10 – Доступ разрешен

В данной статье был рассмотрен пример реализации аутентификации JWT на основе ролей в SpringBoot.

Библиографический список

1. Колойдчук А.В. Программирование инвестиционных и кредитных рисков введение икт с помощью pascal и visual basic // Формирование рыночных отношений в Украине. 2020 №4(227). С. 97-101.
2. Симаков А.А. Java tracer. Программное средство для трассировки java программ // Заметки по информатике и математике. 2019. №3. С. 51-57.
3. Иванов А.Н. Разработка приложения с использованием веб-технологий java // Ученые заметки тогу. 2020. №4. С. 261-266.
4. Великов В.П., Добрева К.С. Генератор из диаграмм классов java в исходный код java // Информационные системы и технологии: управление и

безопасность. 2014. №3. С. 14-23.
5. Шейн А.А. Генератор исходного кода на языке java по описанию бортовых компонентов decode // Современные проблемы науки и образования. 2016 №8. С. 18-25.