# RESEARCH ON JWT AUTHENTICATION APPROACHES IN SPRING BOOT FOR REST API

**Nurdoolot Kalybekov** - nurdoolot.kalybekov@alatoo.edu.kg

Gulzada Esenalieva
**Ala-Too International University**

## ABSTRACT

JWT (JSON Web Token) is one of the most widely used methods for securing REST APIs in modern web applications. This article explores the implementation of JWT authentication in Spring Boot applications and discusses best practices for enhancing the security of REST APIs. Key areas include token generation, validation, storage, and the integration of Spring Security with JWT.

**Keywords**: Spring Boot, REST API, JWT, authentication, authorization, Spring Security, token validation, secure REST API.

# ИССЛЕДОВАНИЕ JWT АУТЕНТИФИКАЦИИ В SPRING BOOT ДЛЯ REST API

**Нурдоолот** Калыбеков - nurdoolot.kalybekov@alatoo.edu.kg

Gulzada Esenalieva
**Международный Университет Ала-Тоо**

## АННОТАЦИЯ

JWT (JSON Web Token) является одним из наиболее популярных способов обеспечения безопасности REST API в современных веб-приложениях. В статье рассматривается реализация JWT аутентификации в приложениях Spring Boot и обсуждаются лучшие практики для повышения безопасности REST API. Основное внимание уделено генерации токенов, их валидации, хранению, а также интеграции Spring Security с JWT.

**Ключевые слова**: Spring Boot, REST API, JWT, аутентификация, авторизация, Spring Security, проверка токенов, безопасный REST API.

# JWT ТИРКЕМЕДЕГИ АУТЕНТИФИКАЦИЯНЫ ИЗИЛДӨӨ

**Нурдөөлөт Калыбеков** - nurdoolot.kalybekov@alatoo.edu.kg

Gulzada Esenalieva
**Эл Аралык Ала-Тоо Университети**

## АННОТАЦИЯ

JWT (JSON Web Token) REST API үчүн коопсуздукту камсыздоонун эң популярдуу ыкмаларынын бири. Бул макалада Spring Boot тиркемелеринде JWT аутентификациясынын ишке ашырылышы жана REST API'лердин коопсуздугун жакшыртуу үчүн эң мыкты практикалар талкууланат. Негизги көңүл токендерди түзүү, текшерүү, сактоо жана JWT менен Spring Security'нин интеграциясына бурулат.

**Ачкыч сөздөр**: Spring Boot, REST API, JWT, аутентификация, авторизация, Spring Security, токенди текшерүү, коопсуз REST API.

## 1. INTRODUCTION

Modern web applications require robust authentication and authorization mechanisms. JSON Web Token (JWT) has emerged as a lightweight, stateless solution for securing REST APIs. JWT ensures that each client request to the server is authenticated and authorized without relying on server-side sessions.

This research investigates the implementation of JWT-based authentication in Spring Boot, focusing on the generation of tokens, their validation, and integration with Spring Security. The study explores essential security aspects, such as token expiration, refresh tokens, and secure storage.

## 2. MAIN PART

### 2.1 JWT Authentication in Spring Boot

JWT comprises three parts:

1. **Header**: Contains the token type and hashing algorithm (e.g., HS256).
2. **Payload**: Includes user claims, such as user ID and roles.

3. **Signature**: A hashed value combining the header, payload, and a secret key for verification.

When a user logs into a system, Spring Security authenticates the credentials. Upon successful authentication, the application generates a JWT token and returns it to the client. The client uses this token in the `Authorization` header for subsequent requests.

**2.2 Token Generation Example**

Below is a sample code snippet for generating JWT tokens in Spring Boot:

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.stereotype.Service;

import java.util.Date;

@Service
public class JwtService {
    private static final String SECRET_KEY = "your_secret_key";

    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() +
1000 * 60 * 60 * 10)) // 10 hours
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }
}
```

---

**2.3 Token Validation**

Validating a JWT ensures its authenticity and expiration status. In Spring Boot, a filter is commonly used for this purpose:

```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import org.springframework.stereotype.Service;
```

```java
@Service
public class JwtValidationService {
    private static final String SECRET_KEY = "your_secret_key";

    public Claims validateToken(String token) {
        return Jwts.parser()
                .setSigningKey(SECRET_KEY)
                .parseClaimsJws(token)
                .getBody();
    }
}
```

**2.4 Spring Security Integration**

Integrating Spring Security with JWT involves configuring security filters and managing user roles:

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.configur
ation.AuthenticationConfiguration;
import
org.springframework.security.config.annotation.web.builders.HttpSecuri
ty;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        return http
                .csrf().disable()
                .authorizeHttpRequests(auth -> auth
                    .requestMatchers("/auth/**").permitAll()
                    .anyRequest().authenticated()
                )
                .build();
    }

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

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration config) throws
Exception {
        return config.getAuthenticationManager();
    }
}
```

---

### 2.5 Best Practices

1. **Token Expiration**: Always set an expiration time for tokens to limit exposure.
2. **Refresh Tokens**: Use refresh tokens for long-lived sessions to enhance security.
3. **Secure Storage**: Store tokens securely, preferably in HTTP-only cookies or encrypted local storage.
4. **Encryption**: Use strong encryption algorithms like HS512 or RS256.

---

## 3. CONCLUSION

JWT authentication in Spring Boot offers a robust and scalable solution for securing REST APIs. By leveraging the built-in capabilities of Spring Security and following best practices, developers can create secure applications that protect user data and prevent unauthorized access. Proper token management, including expiration and validation, is critical to ensuring the overall security of REST APIs.

## 4. REFERENCES

1. GeeksforGeeks. "Introduction to JWT in Spring Security." https://www.geeksforgeeks.org/jwt-spring-security/
2. Spring. "Spring Security Overview." https://spring.io/projects/spring-security
3. JWT.io. "Introduction to JSON Web Tokens." https://jwt.io/
4. Baeldung. "Guide to JSON Web Tokens in Spring Security." https://www.baeldung.com/spring-security-jwt
5. Additional informations were obtained from https://scholar.google.ru/scholar?hl=ru&as_sdt=0%2C5&q=Exploring+Approaches+to+Implementing+JWT+Authentication+in+Spring+Security&btnG=