# Data Security Approaches in Spring Boot for REST API

Saikal Batyrbekova and Gulzada Esenalieva

January 6, 2025

# RESEARCH ON DATA SECURITY APPROACHES IN SPRING BOOT FOR REST API

***Batyrbekova Saikal* - *saikal.batyrbekova@alatoo.edu.kg***

***Gulzada Esenalieva* - *gulzada.esenalieva@alatoo.edu.kg***

*Ala-Too International University*

## ABSTRACT

Spring Boot offers a wealth of tools for securing web applications. This article takes a detailed look at Spring Boot's built-in security and authentication mechanisms and how they can be used to secure REST APIs. Particular attention is paid to aspects such as data encryption, CSRF protection and context security.

**Keywords:** Security, authentication, Spring Boot, REST API, data protection, encryption, CSRF, context security

# ИССЛЕДОВАНИЕ ПОДХОДОВ К ОБЕСПЕЧЕНИЮ БЕЗОПАСНОСТИ ДАННЫХ В SPRING BOOT ДЛЯ REST API

***Батырбекова Сайкал* - *saikal.batyrbekova@alatoo.edu.kg***

***Гульзада Эсеналиева* - *gulzada.esenalieva@alatoo.edu.kg***

*Международный Университет Ала-Тоо*

## АННОТАЦИЯ

Spring Boot предлагает большой инструментарий для обеспечения безопасности веб-приложений. В статье детально рассматриваются встроенные механизмы безопасности и аутентификации Spring Boot, а также их применение для защиты REST API. Особое внимание уделяется таким аспектам, как шифрование данных, защита от CSRF и контекстная безопасность.

**Ключевые слова:** безопасность, аутентификация, Spring Boot, REST API, защита данных, шифрование, CSRF, контекстная безопасность

# REST API ҮЧҮН SPRING BOOT ТИРКЕМЕДЕИГИ КООПСУЗДУК ЫКМАЛАРЫН ИЗИЛДӨӨ

***Батырбекова Сайкал -*** *saikal.batyrbekova@alatoo.edu.kg*

***Гульзада Эсеналиева -*** *gulzada.esenalieva@alatoo.edu.kg*

*Эл Аралык Ала-Тоо Университети*

## АННОТАЦИЯ

Spring Boot веб-тиркемелерди коргоо үчүн көптөгөн куралдарды сунуштайт. Бул макалада Spring Boot'тун орнотулган коопсуздук жана аутентификация механизмдери жана аларды REST API'лерди коргоо үчүн кантип колдонсо болорун деталдуу карап чыгат. Маалыматтарды шифрлөө, CSRF коргоо жана контексттик коопсуздук сыяктуу аспектилерге өзгөчө көңүл бурулат.

**Ачкыч сөздөр:** коопсуздук, аутентификация, Spring Boot, REST API, маалыматтарды коргоо, шифрлөө, CSRF, контексттик коопсуздук

## 1. INTRODUCTION

Ensuring security in modern web applications has become one of the most pressing tasks for developers. With the growing number of cyberattacks and data breaches, safeguarding user information and data confidentiality is increasingly important. Spring Boot is a platform for creating new Spring applications, offering numerous built-in security and authentication mechanisms that help developers create secure applications. The security and authentication mechanism in Spring Boot is primarily based on the Spring Security platform, a core component of the Spring ecosystem widely used to provide security features.

Spring Security provides several key concepts: authentication, the process of verifying a user’s identity; authorization, the process of determining whether the user has permission to access a specific resource; and session management, which refers to managing user sessions within an application. These concepts work together to form the security and authentication mechanism in Spring Boot, where authentication identifies the user, authorization confirms their access permissions, and session management oversees user sessions within the application.

Data security in modern information systems is of increasing significance. With the rise of cyberattacks and information leaks, protecting user data privacy has become one of the top priorities for developers and system administrators. REST API, one of the most popular methods of interaction between client and server applications, requires special attention in terms of security. Utilizing Spring Boot, a powerful framework for creating microservices, significantly simplifies the process of developing secure REST APIs.

The purpose of this research is to examine various approaches to ensuring data security in REST APIs developed using the Spring Boot framework. This study will analyze existing security mechanisms, such as data encryption, protection against Cross-Site Request Forgery (CSRF), and the implementation of contextual security. To achieve this, the study will investigate basic security principles in REST API development, explore Spring Boot's security capabilities, analyze encryption methods used in Spring Boot, study CSRF protection mechanisms and their implementation in Spring Security, consider contextual security and its application in Spring Boot for managing resource access, and compare the effectiveness of different approaches to securing data in REST APIs.

The object of this research is the process of ensuring data security in REST APIs developed with Spring Boot, while the subject of this research includes the data security methods and tools, such as encryption, CSRF protection, and contextual security implemented in Spring Boot.

This study will help identify optimal methods for ensuring security in various REST API scenarios, develop recommendations for enhancing the security level of applications based on Spring Boot, and foster an understanding of modern trends in web application data security. In summary, the results of this study may be valuable for developers, architects, and information security specialists involved in creating and maintaining REST APIs on the Spring Boot platform.

## 2. MAIN PART

No matter what our settings are, the server side has to do two important things anyway - authentication and authorization. So, what are authentication and authorization in simple words? Authentication is the process of establishing the identity of the user, our web application tries to understand if the user is an imposter and if he is really who he claims to be. Authorization in turn determines what actions the user can perform in the web application. Before getting access to perform these actions and get the necessary resources - the first thing you need to do is to pass authentication.

Let's see how authentication in Spring Security takes place in the context of REST API. First, we need to get the user's credentials. The user enters their username and password into the registration form on the web page or passes them in the HTTP request header. Typically, the *UsernamePasswordAuthenticationFilter* in Spring Security is responsible for extracting the user's credentials from the HTTP request to the REST API. It then needs to compare them to the actual username and password stored somewhere in the database. This is done by *AuthenticationManager* in the *authenticate()* method. And the last step is to generate a JWT token, of course, after successful authentication. Talking about the authorization process in the REST API, as I mentioned earlier, after successful authentication, a JWT token is generated, which is like a pass ticket that allows the user to perform actions in the web application. After receiving the token, the client sends the token in every subsequent request. For example, a request to send a message, edit a profile or purchase some product.

Authentication and authorization in REST APIs, of course, play a key role in data protection, but they cannot guarantee data security if the data itself remains vulnerable. It's no secret that storing sensitive information unencrypted is very risky. That's why data protection and encryption become indispensable in this case.

Spring Boot recommends using standard encryption libraries such as JCE (Java Cryptography Extension) to encrypt passwords, tokens and other sensitive data. For password encryption, you should use BCrypt, which is built into Spring Security and provides strong protection against password attacks.

The Java Cryptography Extension works well as a set of algorithms and is already built into Java, making it easy to use its classes and methods without a separate installation. Here's what an encrypted password looks like in the JCE extension:

```
Original Password: Ala-Too2025
Encrypted Password: S7mv+Q4J8DyVlIswQjQbzw==
```

When a user enters their password to register or change it, Spring Security uses the BCrypt algorithm to create a hash of the password. Hashing is a process that converts a password into a fixed length string that cannot be converted back to the original password. Java Spring Boot Code Snippet:

```
package com.example.Memories_App.config;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
```

```
@RequiredArgsConstructor
public class ApplicationConfig {
    private final UserRepository userRepository;
    @Bean
    public UserDetailsService userDetailsService() {
        return email -
> userRepository.findByEmail(email).orElseThrow(() -> new CustomException("User
not found: ", HttpStatus.NOT_FOUND));
    }
    @Bean    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration
config) throws Exception {
        return config.getAuthenticationManager();
    }
}
```

While encryption and data protection provide security at the storage and transmission level, it is important to consider threats that can occur at the user-application interaction level. One such threat is CSRF.

Cross-site CSRF request forgery is an attack in which an attacker or a hacker forces a user to perform an unwanted action on a site to which they are authenticated. In the case of REST APIs, defense against CSRF attacks becomes especially important because tokens and sessions can be vulnerable to these types of attacks. I would like to tell you how these unfortunate CSRF attacks happen. Imagine, you log in to an online bank, and then you go to a website with an advertisement that says you've won 100,000 soms. You click on the link and it secretly sends a request to transfer money from your account to the attacker's account. Since you are still authorized, the bank performs the transfer without suspecting fraud. This is a CSRF attack, where the attacker uses your active session for their own egocentric purposes. It is important to connect CSRF protection in Spring Security to prevent such attacks. Spring Security automatically adds a CSRF token to all forms and validates it with every request, which prevents fake requests from being made.

CSRF protection is based on preventing spoofed requests, but it alone does not solve the security problem if the user's session is not properly secured. After all, if an attacker gains access to the session, he can perform actions on behalf of the user, bypassing even CSRF protection. Therefore, session management is an important aspect of security.

In Spring Security it is possible to customize such parameters as session lifetime: each session has an expiration date, which can be limited to inactivity time (if the user does not make requests for 30 minutes, the session is closed) or a certain duration (the session ends after 12 hours), the number of concurrent sessions per user (a user can log in to their account on a laptop and simultaneously on a smartphone). For this purpose, the *sessionManagement()* method in the REST API in Spring Security is used:

```
@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception
{

    return http

            .csrf(csrf -> csrf.disable())

            .sessionManagement(session -> session

                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)

                .maximumSessions(1)

                .maxSessionsPreventsLogin(true)

                .and()

                .sessionFixation().migrateSession()            )

             .authorizeHttpRequests(auth -> auth

                .requestMatchers("/admin/**").hasRole("ADMIN")

                .anyRequest().authenticated()

            )

         .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)

         .build();

}
```

Security is a process that requires constant attention and improvement. By following the best security practices in Spring security in REST API, a developer can significantly reduce risks and protect their application from a multitude of threats. Utilizing Spring Security's built-in tools, regularly updating dependencies, encrypting data, and being mindful of session management are important steps towards creating a safe and secure application.

This paper looked into security measures when developing REST APIs using Spring Boot such as identifying and verifying users, roles and policies, database encryption, protection against CSRF attacks and user sessions among others. Spring boot together with its friend, Spring security has provided great features in building applications that are secure and confidentiality as well as integrity of information is achieved. Authentication and authorization are one of the most important elements of a security model; these are achieved through the use of passwords and user IDs stored in JSON web tokens (JWT). Encryption of sensitive data is one of the most important goals of IT security, projects that help achieve this goal such as BCrypt and JCE are considered. Defense against malicious attacks is the main goal of CSRF protection, and management of user authentication sessions and variables is designed to enable or disable user sessions.

## 3. CONCLUSION

In conclusion, securing REST APIs in Spring Boot requires the application of several built-in mechanisms for security such as securing the channel and preventing attacks. Applying encryption, CSRF protection, and proper session management tools can greatly help improve the security of the developers' applications. It is important to note that even basic maintenance such as working on dependencies and enforcing more sophisticated policies such as OAuth2 is critical in the war against new threats. Therefore, by using these techniques, the REST API created by Spring Boot developers are able to protect user information and the overall application's functionality.

## 4. REFERENCES

- Esenalieva, G. A. (n.d.). *Cyber Security in the Education System.* Candidate of pedagogical sciences, lecturer at the Department of Informatics and Computer Engineering, Ala-Too International University.
- A. Khan, R. R. Mekuria and R. Isaev, *"Applying Machine Learning Analysis for Software Quality Test," 2023 International Conference on Code Quality (ICCQ)*
- Токтогулова Г. А., Тороев А. А., & Эсеналиева Г. А. (n.d.). *Безопасность информационных систем.*
- GeeksforGeeks. "CSRF Protection in Spring Security." https://www.geeksforgeeks.org/csrf-protection-in-spring-security/
- Spring. "Spring Security." https://spring.io/projects/spring-security
- CSDN. "Spring Security CSRF Protection Implementation." https://blog.csdn.net/suifengme/article/details/136719602
- Snyk. "CSRF Attack: Understanding and Mitigation." https://learn.snyk.io/lesson/csrf-attack/

- GeeksforGeeks. "Spring Boot: Enhancing Data Security with Column-Level Encryption." https://www.geeksforgeeks.org/spring-boot-enhancing-data-security-column-level-encryption/
- Habr. "Как защититься от CSRF атак в Spring Security." https://habr.com/ru/articles/798921/
- Stack Overflow. "How to Secure REST APIs in Spring Boot Web Application." https://stackoverflow.com/questions/42870489/how-to-secure-rest-apis-in-spring-boot-web-application