Introduction

Radu Andrei Popescu, sn:4603052

Fontys University of Applied Sciences

**Introduction**

Java Spring Boot is a popular framework for building web applications. It includes built-in support for authentication and authorization, which are critical for ensuring the security of web applications. I am currently a student that wants to implement in his Java application authorization and authentication. My main criteria are that it should fit in my level of expertise, the level of security should be average; however, it should provide the necessary functionalities of any authorization tool and to fit in my project requirements. The main question of this study is:

What are the best tools to implement in my project?

The sub-questions are:

What are the strengths, weaknesses, opportunities and threats of each tool?

What is the best way to implement the desired tool in my project?

# Methodology

The study design involves a combination of quantitative literature study and prototyping. The theoretical analysis examines the different tools available at my disposal, such as: Spring Security, OAuth2.0, JSON web Tokens and Apache Shiro. The practical experimentation involves examples of simple Springboot applications from the internet that implement those tools. The data will be analyzed using a SWOT diagram to show the best pick for my project.

# Results

The available tools for my project are the following. Spring Security, a powerful and highly customizable security framework for Java applications. It provides a wide range of authentication and authorization features, including support for OAuth 2.0, OpenID Connect, and SAML. OAuth 2.0, which is a widely adopted authorization framework that allows applications to access resources on behalf of a user. Spring Boot provides built-in support for OAuth 2.0, making it easy to integrate with third-party authentication providers like Google, Facebook, and GitHub. JSON Web Tokens (JWT), which is a JSON-based token format that is commonly used for authentication and authorization. Spring Boot provides support for JWT, making it easy to generate and validate tokens. And lastly Apache Shiro, which is a powerful and flexible security framework for Java applications. It provides a wide range of authentication and authorization features, including support for role-based access control, multi-factor authentication, and more.

With this in mind, we need to look at the strengths, weaknesses, opportunities and threats of each tool to determine which one is the most suitable for my project. I have designed a SWOT diagram below which should provide clarity on the respective tools.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tools** | **Strengths** | **Weaknesses** | **Opportunities** | **Threats** |
| Spring Security | - Widely used and well-documented  - Highly customizable  - Integrates well with Spring components | - Steep learning curve for beginners  - Configuration can be complex for some use cases | - Provides a solid foundation for implementing authentication and authorization  - Support for a wide range of security features | - Competing security frameworks in the market  - Potential vulnerabilities if not implemented correctly |
| OAuth 2.0 | - Widely adopted standard  - Support for third-party authentication providers  - Good for implementing API security | - Can be complex to implement  - Limited support for user management  - Potential security risks if not implemented correctly | - Provides a standardized protocol for securing APIs and third-party authentication  - Can improve user experience by reducing the need for multiple login credentials | - Vulnerabilities if not implemented correctly  - Can be too complex for some use cases |
| JSON Web Tokens (JWT) | - Lightweight and easy to implement  - Good for stateless authentication  - Can be used in distributed environments | - Can lead to performance issues with large tokens  - No built-in token revocation mechanism | - Provides a simple and flexible way to implement token-based authentication  - Can be used to securely transmit information between services | - Vulnerabilities if not implemented correctly  - Potential security risks if tokens are compromised |
| Apache Shiro | - Simple and easy to use  - Good for role-based access control  - Supports multiple authentication mechanisms | - Limited support for third-party authentication providers  - No built-in support for OAuth 2.0 | - Provides a comprehensive security framework for Java applications  - Good for implementing fine-grained access control | - Smaller community compared to Spring Security  - Potential vulnerabilities if not implemented correctly |

# Discussion

After careful analysis of the diagram, JSON Web Token provides the best implementation of the authorization and authentication along with Spring Security. Both have a wide and active community that can answer questions and provide feedback and also are within the level of expertise a university student has. While the Spring Security has a steep learning curve it is backed by its wide community. Both possess vulnerabilities if not implemented correctly, however, looking past this threat, they provide a simple and secure way of security to the application. Now that we have an answer to one of the sub-questions, one still remains.

To answer it, here are the steps to implement Spring Security and JWT in a Java Spring Boot application: Add the necessary dependencies in my project. The dependencies required are spring-boot-starter-security, jjwt, and spring-boot-starter-web. Create a class that extends the WebSecurityConfigurerAdapter class, which provides the configuration for Spring Security. In this class, I can specify the authentication provider, the allowed endpoints, and other security configurations. Create a class that represents the user and implements the UserDetails interface. This class should contain the user’s username, password, and any other relevant information.

Create a class that implements the AuthenticationProvider interface. This class will be used to authenticate the user’s credentials against your user database. Create a login endpoint that accepts a username and password, authenticates the user, and returns a JWT token. Create an endpoint that requires authentication and accepts a JWT token in the Authorization header. This endpoint should verify the JWT token and allow access if the token is valid. Create a class that generates and verifies JWT tokens using the jjwt library. This class should contain the secret key, expiration time, and other configuration settings for my JWT token. Add the @EnableWebSecurity annotation to the main application class to enable Spring Security. Configure Spring Security to use JWT for authentication by adding a filter that intercepts requests and verifies the JWT token. With these steps, I should be able to implement Spring Security and JWT in your Java Spring Boot application.

# Conclusion

This document has discussed the available tools for implementing authorization and authentication in a Java SpringBoot application and their differences, which one would be best suited for my project and what is the best way of implementing it. Overall, JSON Web Token and Spring Security provide the answer to all the questions asked in this research paper due to their well-documented user base, easy to learn for beginners an easily integrable system with the tools that I use..

References

<https://auth0.com/intro-to-iam/what-is-authentication>

<https://www.bu.edu/tech/about/security-resources/bestpractice/auth/>

<https://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#what-is-acegi-security>

https://spring.io/guides/gs/securing-web/

<https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/in-memory.html>

<https://chat.openai.com/>

<https://www.callicoder.com/spring-boot-security-oauth2-social-login-part-1/>