Password Hashing Technique

Introduction:

Storing passwords securely is of utmost importance in any application. In Spring Boot, you can use password hashing techniques to protect user passwords from being compromised. Password hashing ensures that even if the password hashes are exposed, they cannot be easily reversed to obtain the original password.

PasswordEncoder Interface:

- Spring Security provides the ***PasswordEncoder*** interface, a general contractor for encoding and verifying passwords.
- You can implement this interface to encode passwords and choose a specific encryption algorithm, such as BCrypt, Argon2 or SHA-256.
- The PasswordEncoder interface provides methods like "***encode()"*** to encode a password and matches() to verify if a given password matches the encoded value.

MessageDigestPasswordEncoder:

- This password encoder uses the Java MessageDigest class to perform password hashing.
- To hash the passwords, you can choose a specific hashing algorithm, such as MD5, SHA-256, SHA-512, etc.
- It is important to note that MD5 is considered weak for password hashing due to its vulnerability to dictionary attacks.

Argon2PasswordEncoder:

- Argon2 is a memory-hard hashing algorithm that provides resistance against GPU and FPGA attacks.
- Spring Security introduced the Argon2PasswordEncoder class, which allows you to use Argon2 for password encoding.
- Argon2 has different configuration parameters, such as iterations, memory, and parallelism, which can be adjusted to increase the algorithm's security.

BCrypt Password Hashing:

One commonly used password hashing algorithm in Spring Boot is bcrypt. BCrypt is a one-way hashing function that incorporates a salt value and a cost factor to make the process computationally expensive. The salt value ensures that their password hashes will differ even if two users have the same password.

Using BCrypt in Spring Boot:

To utilise BCrypt password hashing in your Spring Boot application, you can follow these steps:

1. **Dependency Configuration:**

   Ensure that your Spring Boot project has the necessary dependencies for password hashing. You can include the spring-security-crypto dependency in your build configuration file (e.g., Maven's pom.xml or Gradle's build. gradle).

2. **Hashing Passwords:**

   In your application, you can hash passwords using the BCryptPasswordEncoder class provided by Spring Security. This class implements the PasswordEncoder interface and allows you to generate password hashes easily. Typically, you'll use this encoder during user registration or password update processes.

Here's an example of how to use BCryptPasswordEncode to hash a password, but you need to implement the logic in your respective service class rather than the Main Class:

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public class PasswordHashingExample {
    public static void main(String[] args) {
        String password = "myPassword123";
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        String hashedPassword = passwordEncoder.encode(password);
        System.out.println("Hashed password: " + hashedPassword);
    }
}
```

Running this code will generate a hashed password similar to *$2a$10$ZoCO6s0iDdGIs1hmLwhrtuK.xb2mtwXe3F56W3Kh5TD53REHIEpK2.*

3. **Verifying Passwords:**

   When a user logs in or authenticates, you must compare the entered password with the stored hashed password. Spring Security provides the matches(CharSequence rawPassword, String encodedPassword) method in BCryptPasswordEncoder to perform this comparison. It will internally handle the salt and cost factor to validate the password.

Here's an example of how to verify a password and implement the same in your respective Service Class :

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public class PasswordVerificationExample {
    public static void main(String[] args) {
        String enteredPassword = "myPassword123";
        String storedHashedPassword =
"$2a$10$ZoCO6s0iDdGIs1hmLwhrtuK.xb2mtwXe3F56W3Kh5TD53REHIEpK2";
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        boolean passwordMatches = passwordEncoder.matches(enteredPassword,
storedHashedPassword);
        System.out.println("Password matches: " + passwordMatches);
    }
}
```

Running this code will output Password matches: true if the entered password matches the stored hashed password.
By using BCrypt password hashing technique in your Spring Boot application, you can enhance the security of user passwords and protect them from unauthorised access.

These are some password encryption techniques commonly used in Spring Boot applications. Choosing a strong and secure password encryption technique is important based on your specific requirements and the level of security needed for your application.

Additional Resources:

- Spring Security Documentation
- Spring Boot Documentation