Password Security with Spring Security Methods and Hash Functions

```
package com.example.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.ldap.authentication.LdapPasswordEncoc

@Configuration
public class SecurityConfig {

    // Bean for BCryptPasswordEncoder
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(); // Secure hashing for passwor
    }

    // Bean for NoOpPasswordEncoder (For demonstration purposes only, not
    @Bean
    public PasswordEncoder noOpPasswordEncoder() {
        return NoOpPasswordEncoder.getInstance(); // No hashing, stores p
    }

    // Bean for LdapPasswordEncoder (Use this in LDAP-based authenticatic
    @Bean
    public PasswordEncoder ldapPasswordEncoder() {
        return new LdapPasswordEncoder(); // Specialized for LDAP authent
    }
}


package com.example.security.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/auth")
public class AuthController {
```

```
    private final PasswordEncoder passwordEncoder;

    @Autowired
    public AuthController(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

    @PostMapping("/register")
    public String registerUser(@RequestParam String username, @RequestPar
        // Hash the password before storing
        String encodedPassword = passwordEncoder.encode(password);
        return "User " + username + " registered with encoded password: "
    }

    @PostMapping("/authenticate")
    public String authenticateUser(@RequestParam String username, @Reques
        // Validate the password against the stored hashed password
        // In a real scenario, you would retrieve the encoded password fr
        String storedPassword = "$2a$10$Dow56cExZOWcQMXkDA2rr.1dqV5Se5ntk

        if (passwordEncoder.matches(password, storedPassword)) {
            return "Authentication successful!";
        }
        return "Authentication failed!";
    }
}
```

Explanation of Password Encoders Used:

- **BCryptPasswordEncoder**: This encoder applies the BCrypt hashing algorithm to passwords and is recommended for securing passwords in a production environment.
- **NoOpPasswordEncoder**: This encoder does not hash passwords and is used for testing or scenarios where plain-text passwords are temporarily acceptable. It should never be used in production as it exposes passwords in clear text.
- **LdapPasswordEncoder**: This encoder is specialized for LDAP-based authentication systems, using the standard LDAP hashing mechanism.