Setting Password Encoder

After implementing password encoding during registration, we need to update our authentication system to verify encoded passwords. The key points:

- Previously, we were using NoOpPasswordEncoder which expects plain text passwords
- Now that our passwords are encoded with BCrypt, we need to use BCryptPasswordEncoder for authentication
- This ensures the system can properly verify passwords during login

👉 Updating the Security Configuration

In our SecurityConfig class, we need to make the following changes:

- Keep the @Configuration and @EnableWebSecurity annotations that mark this as a Spring Security configuration
- The AuthenticationProvider bean must now use BCryptPasswordEncoder instead of NoOpPasswordEncoder
- Set the same strength parameter (12) as we used in the registration process

Example:

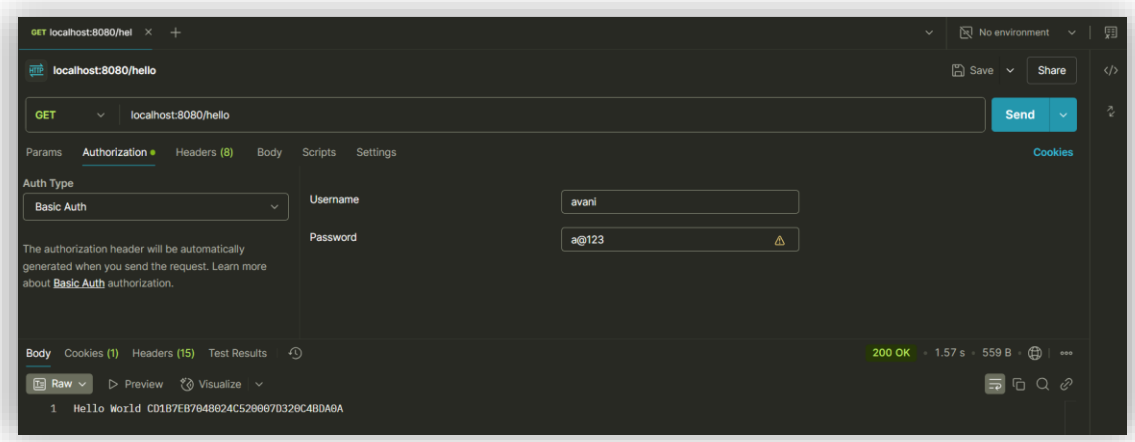
```java
package com.telusko.springsecdemo.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public AuthenticationProvider authProvider() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService);
        provider.setPasswordEncoder(new BCryptPasswordEncoder(12));
        return provider;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http.csrf(customizer -> customizer.disable())
                .authorizeHttpRequests(request -> request.anyRequest().authenticated())
                .httpBasic(Customizer.withDefaults())
                .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        return http.build();
    }
}
```

Output:



- User provides credentials (username and password)
- Spring Security retrieves the user details using UserDetailsService
- The BCryptPasswordEncoder takes the plain text password from the login attempt and applies the same hashing algorithm
- It then compares this hash with the stored hash from the database
- If they match, authentication succeeds