Web Security Config

Path: com.opus.backend.config.WebSecurityConfig.class

This class that extends **WebSecurityConfigurerAdapter**, is used to configure Spring Security settings in a Spring Boot application.

1. **@Configuration**: Indicates that the class is a configuration class that Spring should use to create beans.
2. **@EnableWebSecurity**: Enables Spring Security's web security support, which is required to provide security features.
3. **@EnableGlobalMethodSecurity(prePostEnabled = true)**: Enables method-level security with support for Spring Expression Language (SpEL) expressions.

The class **WebSecurityConfig** contains the following methods:

1. **configure(AuthenticationManagerBuilder auth)**: Configures the **AuthenticationManagerBuilder** to use the **jwtUserDetailsService** for loading user information and the BCrypt password encoder for password encoding and matching.
2. **passwordEncoder()**: A bean definition for a **PasswordEncoder** using the BCrypt hashing algorithm.
3. **authenticationManagerBean()**: Exposes the **AuthenticationManager** from the superclass as a bean that can be used by other components.
4. **configure(HttpSecurity httpSecurity)**: Configures the **HttpSecurity** object to specify the security rules for the application. It disables CSRF protection, permits specific requests without authentication, and requires authentication for all other requests. It also configures exception handling with a custom **JwtAuthenticationEntryPoint**, sets the session creation policy to stateless (no session for storing user state), and adds a **JwtRequestFilter** to validate tokens for every request.

Some key elements of the **configure(HttpSecurity httpSecurity)** method are:

* **httpSecurity.csrf().disable()**: Disables CSRF protection, as it's not needed for this example.
* **authorizeRequests().antMatchers(HttpMethod.OPTIONS,"/\*").permitAll().antMatchers("/authenticate", "/register","/verify","/hello1","/reset\_password","/forget\_password").permitAll()**: Allows unauthenticated access to specific endpoints.
* **anyRequest().authenticated()**: Requires authentication for all other requests.
* **exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint)**: Configures a custom authentication entry point to handle unauthorized access attempts.
* **sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)**: Configures the application to use stateless sessions, meaning the user's state won't be stored in a session.
* **httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class)**: Adds the **JwtRequestFilter** before the **UsernamePasswordAuthenticationFilter** to validate tokens for every request.