PhotoAppApiUsers WebSecurity class

I'll explain this Spring Security configuration code line by line:

java

Copy

@Bean

protected SecurityFilterChain configure(HttpSecurity http) throws Exception {

This method is annotated with @Bean, indicating it will produce a bean to be managed by the Spring container. It configures the SecurityFilterChain, which defines the security rules for the application.

java

Copy

AuthenticationManagerBuilder authenticationManagerBuilder =

http.getSharedObject(AuthenticationManagerBuilder.class);

This line retrieves the shared AuthenticationManagerBuilder from the HttpSecurity object.

java

Copy

authenticationManagerBuilder.userDetailsService(usersService)

.passwordEncoder(bCryptPasswordEncoder);

Here, the AuthenticationManagerBuilder is configured with a custom UserDetailsService (usersService) and a password encoder (bCryptPasswordEncoder).

java

Copy

AuthenticationManager authenticationManager = authenticationManagerBuilder.build();

This builds the AuthenticationManager from the configured AuthenticationManagerBuilder.

java

Copy

AuthenticationFilter authenticationFilter =

new AuthenticationFilter(usersService, environment, authenticationManager);

authenticationFilter.setFilterProcessesUrl(environment.getProperty("login.url.path"));

A custom AuthenticationFilter is created and configured with the login URL path from the environment properties.

java

Copy

http.csrf((csrf) -> csrf.disable());

This disables CSRF protection for the application.

java

Copy

http.authorizeHttpRequests((authz) -> authz

.requestMatchers(new AntPathRequestMatcher("/users/\*\*")).access(

new WebExpressionAuthorizationManager("hasIpAddress('"+environment.getProperty("gateway.ip")+"')"))

.requestMatchers(new AntPathRequestMatcher("/h2-console/\*\*")).permitAll())

This configures authorization rules:

* Requests to "/users/\*\*" are allowed only from a specific IP address (gateway.ip).
* Requests to "/h2-console/\*\*" are permitted for all users.

java

Copy

.addFilter(new AuthorizationFilter(authenticationManager, environment))

.addFilter(authenticationFilter)

These lines add custom AuthorizationFilter and the previously created AuthenticationFilter to the security filter chain.

java

Copy

.authenticationManager(authenticationManager)

This sets the AuthenticationManager for the HttpSecurity.

java

Copy

.sessionManagement((session) -> session

.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

This configures the session management to be stateless, typically used in REST APIs.

java

Copy

http.headers((headers) -> headers.frameOptions((frameOptions) -> frameOptions.sameOrigin()));

This configures frame options to allow frames from the same origin, often used for H2 console access.

java

Copy

return http.build();

Finally, this builds and returns the configured SecurityFilterChain.

This configuration sets up a secure environment with custom authentication and authorization filters, specific access rules, and stateless session management, suitable for a REST API with particular security requirements.