**Cookies and Sessions**

* **Stateful (Cookies/Sessions)**: The server stores information about the client (e.g., session data) and associates it with a session ID. The client sends this session ID (usually in a cookie) with each request, and the server uses it to retrieve the stored information.
* **Stateless (JWT)**: The server does **not store any information** about the client. Instead, all necessary information is encoded in the JWT (JSON Web Token), which the client sends with each request. The server validates the JWT and extracts the required information from it.

Since you’re using **JWT**, your application is **stateless**. This means:

* The server does **not store any session data**.
* The client (e.g., a frontend app or mobile app) sends the JWT in the Authorization header with each request.
* The server validates the JWT and extracts the user’s information (e.g., roles, permissions) from the token itself.

**Why Your Application is Stateless**

* You’re using JWT for authentication.
* You’ve explicitly set the session creation policy to STATELESS in your SecurityFilterChain:

java

Copy

.sessionManagement(sess -> sess

.sessionCreationPolicy(SessionCreationPolicy.STATELESS)

)

* This ensures that no session is created or stored on the server.

**How JWT Works in Your Application**

1. **Login**: The client sends credentials (e.g., username/password) to the server.
2. **Token Generation**: The server validates the credentials and generates a JWT containing user information (e.g., roles, user ID).
3. **Token Storage**: The client stores the JWT (usually in local storage or cookies).
4. **Subsequent Requests**: The client sends the JWT in the Authorization header with each request:

Copy

Authorization: Bearer <JWT>

1. **Token Validation**: The server validates the JWT and extracts user information from it.

**Why You’re Getting a 403 Error**

The 403 Forbidden error occurs because:

1. The root path (/) is not explicitly allowed in your SecurityFilterChain.
2. The RateLimitingFilter might be blocking requests to /.

**Fixing the 403 Error**

**1. Allow the Root Path (/) in SecurityFilterChain**

Update your SecurityFilterChain to explicitly allow access to the root path:

java

Copy

.requestMatchers("/api/auth/\*\*",

"/v3/\*\*", "/swagger-ui/\*\*","/swagger/\*\*","/webjars/\*\*","/tickojet/\*\*", "/").permitAll()

**2. Exclude the Root Path (/) from Rate Limiting**

Update the EXCLUDED\_PATHS set in your RateLimitingFilter to exclude the root path:

java

Copy

private static final Set<String> EXCLUDED\_PATHS = Set.of(

"/v3/api-docs", "/swagger-ui", "/swagger", "/webjars", "/tickojet", "/"

);

**3. Ensure CORS is Configured Correctly**

If the request is coming from a browser, ensure that CORS is properly configured to allow the request:

java

Copy

CorsConfiguration config = new CorsConfiguration();

config.setAllowedOrigins(List.of("\*")); // Allow all origins

config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS")); // Add OPTIONS for preflight requests

config.setAllowedHeaders(List.of("\*"));

config.setAllowCredentials(true); // Allow credentials if needed

return config;

**Final SecurityFilterChain Configuration**

Here’s the updated SecurityFilterChain:

java

Copy

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http, JwtFilter jwtFilter) throws Exception {

http

.cors(cors -> cors.configurationSource(request -> {

CorsConfiguration config = new CorsConfiguration();

config.setAllowedOrigins(List.of("\*")); // Allow all origins

config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS")); // Add OPTIONS for preflight requests

config.setAllowedHeaders(List.of("\*"));

config.setAllowCredentials(true); // Allow credentials if needed

return config;

}))

.csrf(csrf -> csrf.disable()) // Disable CSRF for stateless APIs

.authorizeHttpRequests(auth -> auth

.requestMatchers("/api/auth/\*\*",

"/v3/\*\*", "/swagger-ui/\*\*","/swagger/\*\*","/webjars/\*\*","/tickojet/\*\*", "/").permitAll() // Allow all auth routes, swagger, and root path

.requestMatchers("/api/admin/\*\*").hasRole("ADMIN") // Restrict admin routes

.requestMatchers("/api/user/\*\*").hasRole("USER") // Restrict user routes

.anyRequest().authenticated() // Protect all other endpoints

)

.sessionManagement(sess -> sess

.sessionCreationPolicy(SessionCreationPolicy.STATELESS) // No sessions

)

.authenticationProvider(authenticationProvider())

.addFilterBefore(rateLimitingFilter, BasicAuthenticationFilter.class) // Add rate limit filter

.addFilterBefore(authFilter, UsernamePasswordAuthenticationFilter.class); // Add JWT filter

return http.build();

}

**Testing**

1. Deploy the updated code to Heroku.
2. Access the root path: https://tickojet-8f5e27f79912.herokuapp.com/.
3. If the issue persists, check the Heroku logs for more details:

bash

Copy

heroku logs --tail

**Key Takeaways**

* Your application is **stateless** because you’re using JWT.
* The 403 Forbidden error is caused by the root path (/) not being allowed in your security configuration.
* Fix the issue by allowing the root path in SecurityFilterChain and excluding it from rate limiting.
* Ensure CORS is properly configured to allow requests from the browser.