can you give me code example where i secure rest api with custome authentication

Here is a complete, working example that combines **Securing a REST API** with **Custom Authentication**.

In this scenario, we are securing the API so that:

1. It is **Stateless** (No sessions).
2. It disables **CSRF** (Standard for REST).
3. It uses a **Custom API Key** header (`X-API-KEY`) instead of a username/password.

1. The Custom Token (Data Holder)

This class holds the API key and tracks whether the user is authenticated.

```
import org.springframework.security.authentication.AbstractAuthenticationToken;
import org.springframework.security.core.GrantedAuthority;
import java.util.Collection;

public class ApiKeyAuthenticationToken extends AbstractAuthenticationToken {
    private final String apiKey;

    // Constructor 1: Before Authentication (Unverified)
    public ApiKeyAuthenticationToken(String apiKey) {
        super(null);
        this.apiKey = apiKey;
        setAuthenticated(false);
    }

    // Constructor 2: After Authentication (Verified with Roles)
    public ApiKeyAuthenticationToken(String apiKey, Collection<? extends GrantedAuthority
        super(authorities);
        this.apiKey = apiKey;
        setAuthenticated(true);
    }

    @Override
    public Object getCredentials() { return null; }

    @Override
    public Object getPrincipal() { return apiKey; }
}
```

2. The Filter (The Interceptor)

This looks for the `X-API-KEY` header on every request.

```
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;

public class ApiKeyAuthFilter extends OncePerRequestFilter {

    private final AuthenticationManager authenticationManager;

    public ApiKeyAuthFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse respo
            throws ServletException, IOException {

        // 1. Try to get the API Key from the header
        String requestApiKey = request.getHeader("X-API-KEY");

        // 2. If no key is present, continue (Spring Security will handle 403 later if ne
        if (requestApiKey == null) {
            filterChain.doFilter(request, response);
            return;
        }

        // 3. Create an unauthenticated token
        ApiKeyAuthenticationToken token = new ApiKeyAuthenticationToken(requestApiKey);

        try {
            // 4. Authenticate via the Manager
            Authentication authResult = authenticationManager.authenticate(token);

            // 5. If successful, save to SecurityContext
            SecurityContextHolder.getContext().setAuthentication(authResult);

        } catch (Exception e) {
            // If auth fails, we clear the context just in case
            SecurityContextHolder.clearContext();
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            response.getWriter().write("Invalid API Key");
            return; // Stop execution here
        }

        // 6. Continue the filter chain
        filterChain.doFilter(request, response);
```

```
    }
}
```

3. The Provider (The Logic)

This checks if the API Key is valid. (In a real app, you would check a database).

```
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.stereotype.Component;

@Component
public class ApiKeyAuthenticationProvider implements AuthenticationProvider {

    @Override
    public Authentication authenticate(Authentication authentication) throws Authenticati
        String apiKey = (String) authentication.getPrincipal();

        // --- CUSTOM LOGIC HERE ---
        if ("secret-123".equals(apiKey)) {
            // Valid Key: Return authenticated token with "ROLE_USER"
            return new ApiKeyAuthenticationToken(apiKey, AuthorityUtils.createAuthorityLi
        } else if ("admin-secret".equals(apiKey)) {
            // Admin Key: Return authenticated token with "ROLE_ADMIN"
            return new ApiKeyAuthenticationToken(apiKey, AuthorityUtils.createAuthorityLi
        } else {
            throw new BadCredentialsException("Invalid API Key");
        }
    }

    @Override
    public boolean supports(Class<?> authentication) {
        return ApiKeyAuthenticationToken.class.isAssignableFrom(authentication);
    }
}
```

4. The Configuration (Wiring it all together)

This is where we define the "Secure REST API" rules.

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.Authenticat
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.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilt
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    // 1. Define the AuthenticationManager (wires up your Custom Provider)
    @Bean
    public AuthenticationManager authenticationManager(HttpSecurity http, ApiKeyAuthentic
        return http.getSharedObject(AuthenticationManagerBuilder.class)
                .authenticationProvider(provider)
                .build();
    }

    // 2. Define the Security Filter Chain (Rest API Security Rules)
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http, AuthenticationManager authM
        http
            // Disable CSRF (Not needed for REST/Stateless)
            .csrf(csrf -> csrf.disable())

            // Set Session to Stateless (No JSESSIONID cookies)
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPo

            // Define who can access what
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/public/**").permitAll()     // Open to everyone
                .requestMatchers("/admin/**").hasRole("ADMIN") // Only Admin Key
                .anyRequest().authenticated()                  // Everything else needs a
            )

            // Add our Custom Filter BEFORE the standard login filter
            .addFilterBefore(new ApiKeyAuthFilter(authManager), UsernamePasswordAuthentic

        return http.build();
    }
}
```

5. The Controller (To Test)

```
@RestController
public class TestController {

    @GetMapping("/public/hello")
    public String publicHello() {
        return "Public: Hello World!";
    }

    @GetMapping("/api/data")
    public String privateData() {
        return "Private: You have a valid API Key.";
    }

    @GetMapping("/admin/settings")
    public String adminSettings() {
        return "Admin: Critical Settings.";
```

```
    }
}
```

How to Test (Using Postman or Curl)

1. **Public URL:** `GET /public/hello` → **200 OK** (No key needed).
2. **Private URL (No Key):** `GET /api/data` → **403 Forbidden**.
3. **Private URL (Wrong Key):** `GET /api/data` with `X-API-KEY: wrong` → **401 Unauthorized** (Caught by filter).
4. **Private URL (User Key):** `GET /api/data` with `X-API-KEY: secret-123` → **200 OK**.
5. **Admin URL (User Key):** `GET /admin/settings` with `X-API-KEY: secret-123` → **403 Forbidden** (Valid key, but wrong role).
6. **Admin URL (Admin Key):** `GET /admin/settings` with `X-API-KEY: admin-secret` → **200 OK**.

⁂

1. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg
2. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg
3. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg
4. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg