To make the required headers for HMAC authentication (e.g., X-HMAC-Signature, X-Access-Key, X-Timestamp, X-Nonce) show up in your Swagger UI, you need to configure your OpenAPI documentation generator (typically Springdoc OpenAPI in a Spring Boot application) to include these as security requirements.

This involves defining a custom security scheme and then applying it to your API.

### Steps to Integrate HMAC Headers into Springdoc OpenAPI (Swagger UI)

1. Add Springdoc OpenAPI Dependency:  
   If you haven't already, add the Springdoc OpenAPI dependency to your pom.xml.  
   XML  
   <!-- pom.xml -->  
   <dependency>  
    <groupId>org.springdoc</groupId>  
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
    <version>2.5.0</version> <!-- Use a recent compatible version with Spring Boot 3.x / Spring 6.x -->  
   </dependency>  
     
   *(Note: The version 2.5.0 is generally compatible with Spring Boot 3.x / Spring 6.x. Always check the latest compatibility matrix for Springdoc OpenAPI if you encounter issues).*
2. Create OpenAPI Configuration:  
   You'll create a Spring @Configuration class to define your OpenAPI documentation. In this class, you'll specify the custom headers as security schemes.  
   Java  
   import io.swagger.v3.oas.annotations.OpenAPIDefinition;  
   import io.swagger.v3.oas.annotations.enums.SecuritySchemeIn;  
   import io.swagger.v3.oas.annotations.enums.SecuritySchemeType;  
   import io.swagger.v3.oas.annotations.info.Info;  
   import io.swagger.v3.oas.annotations.security.SecurityRequirement;  
   import io.swagger.v3.oas.annotations.security.SecurityScheme;  
   import io.swagger.v3.oas.annotations.security.SecuritySchemes;  
   import org.springframework.context.annotation.Configuration;  
     
   @Configuration  
   @OpenAPIDefinition(  
    info = @Info(  
    title = "My Spring Boot API with HMAC",  
    version = "1.0",  
    description = "API documentation for HMAC-protected endpoints."  
    ),  
    security = {  
    @SecurityRequirement(name = "X-Access-Key"),  
    @SecurityRequirement(name = "X-Timestamp"),  
    @SecurityRequirement(name = "X-Nonce"),  
    @SecurityRequirement(name = "X-HMAC-Signature")  
    }  
   )  
   @SecuritySchemes(value = {  
    @SecurityScheme(  
    name = "X-Access-Key",  
    type = SecuritySchemeType.APIKEY,  
    in = SecuritySchemeIn.HEADER,  
    description = "Access Key (Client ID) for HMAC Authentication"  
    ),  
    @SecurityScheme(  
    name = "X-Timestamp",  
    type = SecuritySchemeType.APIKEY,  
    in = SecuritySchemeIn.HEADER,  
    description = "Timestamp of the request (Unix milliseconds)"  
    ),  
    @SecurityScheme(  
    name = "X-Nonce",  
    type = SecuritySchemeType.APIKEY,  
    in = SecuritySchemeIn.HEADER,  
    description = "Unique nonce for replay attack prevention"  
    ),  
    @SecurityScheme(  
    name = "X-HMAC-Signature",  
    type = SecuritySchemeType.APIKEY,  
    in = SecuritySchemeIn.HEADER,  
    description = "HMAC-SHA256 signature of the request"  
    )  
   })  
   public class OpenAPIConfig {  
    // This class defines the OpenAPI configuration  
   }

### Explanation of the Annotations:

* **@Configuration**: Marks this as a Spring configuration class.
* **@OpenAPIDefinition**: This annotation provides general information about your API, including title, version, and description. Crucially, it also defines **global security requirements** using the security attribute. By listing the names of your security schemes here, you tell Swagger UI that these requirements apply to all API operations by default.
* **@SecuritySchemes**: This annotation is a container for multiple @SecurityScheme definitions.
* **@SecurityScheme**: This is where you define each of your custom HMAC headers as an API Key security scheme:
  + name: This is the exact name of the HTTP header (e.g., X-Access-Key). This name will be used in the UI for input.
  + type = SecuritySchemeType.APIKEY: Indicates that this is an API key used for authentication.
  + in = SecuritySchemeIn.HEADER: Specifies that the API key is passed in the request header.
  + description: Provides a helpful description for users in the Swagger UI.
* **@SecurityRequirement**: This annotation references a previously defined @SecurityScheme by its name. When used at the @OpenAPIDefinition level, it means that *all* endpoints defined in this API require *all* listed security schemes.

### How it will look in Swagger UI:

After adding this configuration and running your Spring Boot application, when you navigate to http://localhost:8080/userprofile/swagger-ui.html (adjust for your context path), you will see:

1. A "Authorize" button (or a padlock icon next to operations).
2. Clicking "Authorize" will open a dialog box with input fields for X-Access-Key, X-Timestamp, X-Nonce, and X-HMAC-Signature.
3. When you execute an API request from Swagger UI after filling these fields, the UI will automatically add these headers to your outgoing request.

### Applying to Specific Endpoints (Optional)

If only *some* of your endpoints require HMAC authentication, or if some require a *subset* of these headers, you can apply @SecurityRequirement at the controller class level or even individual method level:

Java

import io.swagger.v3.oas.annotations.security.SecurityRequirement;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
// Apply security requirements to all methods in this controller  
@SecurityRequirement(name = "X-Access-Key")  
@SecurityRequirement(name = "X-Timestamp")  
@SecurityRequirement(name = "X-Nonce")  
@SecurityRequirement(name = "X-HMAC-Signature")  
public class MyApiController {  
  
 @GetMapping("/api/data")  
 public String getProtectedData() {  
 return "This is protected data!";  
 }  
  
 // You can override/remove requirements at method level if needed  
 // @Operation(security = @SecurityRequirement(name = "SomeOtherScheme"))  
 // @GetMapping("/api/another-data")  
 // public String getAnotherData() {  
 // return "Another protected data!";  
 // }  
}

By default, the global @OpenAPIDefinition security requirements will apply to all operations. If you add @SecurityRequirement to a specific controller or method, it will *add* to or *override* the global requirements depending on how you structure it. For your HMAC case, the global definition is usually sufficient unless you have very specific mixed authentication schemes.

**Sources**

1. <https://github.com/Andrea2407-m/lufthansaBeChallange>

2. <https://git.rack.farm/fhnw/flobue/access2trains/access2trains-backend/-/blob/07f2265cb82e7ca6bd0573e7267eadb53b14f9a7/src/main/kotlin/ch/fhnw/oop2/access2trains/web/ConnectionController.kt>