

REST API Security

API Keys and Java Web Tokens

Guadalupe Ortiz Bellot

Computer Science and Engineering Department

Contents

1. API Keys
2. JWT
3. JWE

Contents

1. API Keys
2. JWT
3. JWE

1. API KEYS

Introduction (i)

- When we need to authenticate from another application without accessing the data hosted in the application.
- How API Keys work:
- An API key assigns a unique 128-bit value to a user of the RESTful service.
- This API key is associated to a user and maintained in a datastore.
- The RESTful service then references this datastore together with the service id (optional) and API-KEY with each incoming request.
- Once the incoming requests are validated, access is either granted or denied to the specified end-point.

1. API KEYS

Introduction (ii)

Advantages

- No need to transmit user password during authorization between client and server
- API-KEYs are faster than Digest Authentication as only one call is needed as opposed to 2 calls for every request
- API-KEYs works well with TLS/SSL and should be included in the header for added protection and to prevent the values from being captured in the logs.

1. API KEYS

Where to place the api key

- API-KEY / Service ID Required in Header
- API-KEY and Service ID should be included in the HTTP Headers (`@HeaderParam`) instead of HTTP Parameters via (`@QueryParam`).
- This is especially important when using TLS/SSL as it will guarantee that request data is encrypted end to end and prevent man in the middle attacks.

1. API KEYS

Implementation (i)

- Create a User class with 3 private attributes and their corresponding set/get methods: user, password, apikey
- Optionally create database and table for users
- Alternatively create a static structure in the Hello class

1. API KEYS

Implementation (ii)

With database:

- Include mysql driver jar in the project lib folder (already included)
- Create User class (with attributes userID, password and apiKey and the corresponding get and set)
- Create DBAccess class with methods to create/update/obtain/delete users

apikeysDB - Schema									
apiKeys - Table									
Table Name:		Schema: apikeysDB							
Column Name		Datatype		PK	NN	UQ	B	UN	ZF
userID		VARCHAR(10)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
password		VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
apiKey		VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
		Default/Expression		AI	G				

With static structure:

```
private static Map<String, User> myUserDB = new HashMap<>();
```

1. API KEYS

Implementation (iii)

New user obtain an apikey:

```
@POST  
@Path("/apikey")  
@Consumes(MediaType.APPLICATION_JSON)  
@Produces(MediaType.TEXT_PLAIN)
```

```
public String getApiKey(User myUser)  
{  
    UUID apikey = UUID.randomUUID();  
    User newUser = new User();  
    newUser.setUser(myUser.getUser());  
    newUser.setPassword(myUser.getPassword());  
    newUser.setApiKey(apikey.toString());  
    myUserDB.put(myUser.getUser(), newUser);  
    return apikey.toString();  
}
```

1. API KEYS

Implementation (iv)

- Access with the apikey
- You can add password verification

```
@POST  
@Path("/testapikey1")  
@Consumes(MediaType.APPLICATION_JSON)  
@Produces(MediaType.TEXT_PLAIN)
```

```
public String testing (User myUser, @HeaderParam("apikey") String  
apikey)  
{  
    if (myUserDB.containsKey(myUser.getUser())) {  
        if ((myUserDB.get(myUser.getUser()).getApiKey()).equals(apikey)) {  
            return "GRANTED";}  
    }  
    return "Denied";  
}
```

1. API KEYS

Testing (i)

- Obtain Apikey (i)

The screenshot shows the Postman interface for a POST request to `http://localhost:8080/HeloWorld/demo/hello/apikey`. The 'Authorization' tab is selected, showing 'Digest Auth' as the type. A note says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' Below are fields for 'Username' (restUser2) and 'Password' (redacted). A 'Show Password' checkbox is unchecked.

The screenshot shows the Postman interface for the same POST request. The 'Headers' tab is selected, showing a table with one row: 'Content-Type' set to 'application/json'. Other headers listed as '8 hidden' are not visible in this view.

KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
Content-Type	application/json			
Key	Value	Description		

1. API KEYS

Testing (ii)

- Obtain Apikey (ii)
- Copy the api key obtained.

The screenshot shows the Postman application interface. At the top, it displays a POST request to the URL `http://localhost:8080/HeloWorld/demo/hello/apikey`. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "user": "username",  
3   "password": "password"  
4 }
```

Below the request, the response is shown under the 'Body' tab. The status is 200 OK, with a response time of 10 ms and a size of 186 B. The response body contains a single line of text: `78593836-6e08-46ee-9e80-c78cfcc43ebcd`.

1. API KEYS

Testing (iii)

- Validateapikey (i)
- Paste the copied apikey as a new header parameter.

The screenshot shows the Postman interface for a POST request. The URL is `http://localhost:8080/HelоШоД/demо/hello/testApiKey1`. The 'Auth' tab is selected, showing 'Digest Auth' as the type. A warning message states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' Below this, fields for 'Username' (restUser2) and 'Password' (redacted) are shown, with a 'Show Password' checkbox. A note indicates that the authorization header will be automatically generated when the request is sent.

The screenshot shows the 'Headers' tab for the same POST request. There are two header entries: 'Content-Type' set to 'application/json' and 'apikey' set to '78593836-6e08-46ee-9e80-c78cf4...'. The 'apikey' value is redacted. A table below shows columns for Key, Value, Description, and Presets.

Key	Description	Presets
Content-Type	application/json	
apikey	78593836-6e08-46ee-9e80-c78cf4...	

1. API KEYS

Testing (iv)

- Validate apikey (ii)

The screenshot shows the Postman application interface. At the top, there is a header bar with the method "POST", the URL "http://localhost:8080/HeloWorld/demo/hello/testApiKey1", a "Send" button, and dropdown menus for "Params", "Auth" (selected), "Headers (10)", "Body" (selected), "Pre-req.", "Tests", and "Settings". Below the header, there are tabs for "Cookies" and "Beautify". The "Body" tab is selected, showing a JSON payload:

```
1 {  
2   "user": "username"  
3 }  
4 }
```

Below the body, there are tabs for "Body" (selected), "Cookies", "Headers (5)", and "Test Results". The "Test Results" tab shows a response with status 200 OK, 9 ms, 156 B, and a "Save Response" button. At the bottom, there are buttons for "Pretty", "Raw", "Preview", "Visualize", "Text" (with a dropdown arrow), and a copy icon. The response body is displayed as:

```
1 GRANTED
```

Contents

1. API Keys
2. **JWT**
3. JWE
4. References

JSON WEB TOKENS (JWT)

Introduction

- Authentication through a signed token which can be verified by application servers.
- JSON web tokens contain information that is unique for a user.
- To ensure that the token has not been altered the token contains a cryptographically encrypted digital signature.

JSON WEB TOKENS (JWT)

Structure

- JWT Tokens are encoded with base64 and signed with SHA-256.
- They are composed of a header, a payload and a signature.
 - Header: it contains the key id or jwt type and the hashing algorithm.
 - Payload: it contains the claims (registered (<http://www.iana.org/assignments/jwt/jwt.xhtml>), public and private).
 - Signature: it is composed of a base64 encoded header and payload making a hash using some type of Message Authentication Code (MAC).

JSON WEB TOKENS (JWT)

Communication Flows

1. Initial request with username and password
2. Response with 200 status code and JSON Web Token
3. Following requests include the JWT in the header
4. Responses with 200 status code and response

JSON WEB TOKENS (JWT)

Implementation (i)

- Include the libraries (already included)
- Create a new user type or you can reuse the one you already have
- Create a static JSON Web Key
static JsonWebKey myJwk = null;
- Create a method to *authenticateCredentials*
- Create a method to *testJWT*
- Pay attention to the new imports required from ***org.jose4j*** and ***com.fasterxml.jackson***

More info about public claims: <https://www.iana.org/assignments/jwt/jwt.xhtml>

JSON WEB TOKENS (JWT)

Authenticate credentials (i)

```
@Path("/authenticateJWT")
@GET
@Produces(MediaType.APPLICATION_JSON)

public Response
authenticateCredentials(@HeaderParam("username")
String username,
    @HeaderParam("password") String password)
throws JsonGenerationException,
JsonMappingException, IOException {
```

```
User user = new User();
user.setUser(username);
user.setPassword(password);

RsaJsonWebKey jwk = null;
try {
    jwk = RsaJwkGenerator.generateJwk(2048);
    jwk.setKeyId("1");
    myJwk=jwk;
} catch (JoseException e) { e.printStackTrace(); }
```

JSON WEB TOKENS (JWT)

Authenticate credentials (ii)

```
JwtClaims claims = new JwtClaims();
claims.setIssuer("uca");
claims.setExpirationTimeMinutesInTheFuture(10);
claims.setGeneratedJwtId();
claims.setIssuedAtToNow();
claims.setNotBeforeMinutesInThePast(2);
claims.setSubject(user.getUser());
claims.setStringListClaim("roles", "restUser2");
JsonWebSignature jws = new JsonWebSignature();

jws.setPayload(claims.toJson());
jws.setKeyIdHeaderValue(jwk.getKeyId());
jws.setKey(jwk.getPrivateKey());
jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.RSA_USING_SHA256);

String jwt = null;
try { jwt = jws.getCompactSerialization();
} catch (JoseException e) {System.out.println (e);}
user.setApikey(jwt); //SET TOKEN
return Response.status(200).entity(jwt).build();
```

JSON WEB TOKENS (JWT)

Test JWT (i)

```
@POST  
@Path("/testJWT")  
@Consumes(MediaType.TEXT_PLAIN)  
@Produces(MediaType.TEXT_PLAIN)  
  
public Response testJWT  
(@HeaderParam("token") String token, String  
myName)  
  
    throws JsonGenerationException,  
    JsonMappingException, IOException {
```

```
JsonWebKey jwk = myJwk;  
// Validate Token's authenticity and check claims  
  
JwtConsumer jwtConsumer = new JwtConsumerBuilder()  
    .setRequireExpirationTime()  
    .setAllowedClockSkewInSeconds(30)  
    .setRequireSubject()  
    .setExpectedIssuer("uca")  
    .setVerificationKey(jwk.getKey())  
    .build();
```

JSON WEB TOKENS (JWT)

Test JWT (ii)

```
try {  
    // Validate the JWT and process it to the Claims  
    JwtClaims jwtClaims = jwtConsumer.processToClaims(token);  
    System.out.println("JWT validation succeeded! " + jwtClaims);  
} catch (InvalidJwtException e) {  
    return Response.status(Status.FORBIDDEN.getStatusCode()).entity("Forbidden").build();  
}  
  
String sayHello="Hello "+myName;  
return Response.status(200).entity(sayHello).build();  
}
```

JSON WEB TOKENS (JWT)

Testing JWT (i)

- Get authentication credentials
- Copy the obtained token

The screenshot shows the POSTMAN API client interface. At the top, there is a header bar with 'GET' selected, a URL field containing 'http://localhost:8080/HeloWorld/demo/hello/authenticateJWT', a 'Send' button, and a dropdown menu. Below the header are tabs for 'Params', 'Auth', 'Headers (8)', 'Body', 'Pre-req.', 'Tests', and 'Settings'. The 'Headers' tab is currently active, showing two checked fields: 'username' and 'password'. A table below lists these headers with their values. In the bottom section, there are tabs for 'Body', 'Cookies', 'Headers (5)', and 'Test Results'. The 'Test Results' tab is active, displaying a response status of '200 OK', a duration of '366 ms', and a size of '714 B'. It also includes a 'Save Response' button and a preview area. The preview area shows a JSON object with a single key-value pair: 'eyJraWQiOiIxIiwiYWxnIjoiUlMyNTYifQ.' followed by a long string of encoded data.

KEY	VALUE	DESCRIP	ooo	Bulk Edit	Presets
<input checked="" type="checkbox"/> username	username				
<input checked="" type="checkbox"/> password	password				
Key	Value	Description			

1 eyJraWQiOiIxIiwiYWxnIjoiUlMyNTYifQ.
eyJpc3MiOiJ1Y2EiLCJleHAiOjE2MTQ4NTcxMzEsImp0aSI6IndjcHFTYU1YRVVvYVH1LTWpMRzdiMVEiLCJpYXQiOjE2MTQ4NTY1MzE
sIm5iZiI6MTYxNDg1NjQxMSwic3ViIjoidXNlcm5hbWUiLCJyb2x1cyI6WyJyZXN0VXN1cjIiXX0.
NQ4viJNeRYLxG0bpNK1k9W2RLjtA2aTY4qiEJBh-So6Z3fzsEUREcrgLVeXnfQao8kEJQUGAHV9i4v2Mkz59RBQ8-s-_6uCSGxRGp1
m0hk5X1stp23B0L0jsk98tv4nTMa4_T6nKpLYu7H1Pch04RNc3FJcrM3LPRzsKBf73Upc3EBnnunTxj-Rm9RKQbS5--PHh6p8fDlo9_
Wlq_ta3vGLJmr09s2tICXb16oayzQ43UN8nFLEpKGsxHBI1ycTOSTp6zmuwZrr1WuYKsAW3Eg8NqPVTPfpDxXKg1k1eNUGz2pUziIkM
3dcG_cAgDccMLdnrsB1-kohbdVq-aYMmA

JSON WEB TOKENS (JWT)

Testing JWT (ii)

- Authenticate with JWT(i):

The screenshot shows the Postman interface for a POST request to `http://localhost:8080/HelоАWorld/demo/hello/testJWT`. The 'Authorization' tab is selected. A warning message in a callout box states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' Below the tab, there are fields for 'Username' (set to `restUser2`) and 'Password' (redacted). A 'Show Password' checkbox is also present.

- Paste the obtained token in a new header param

The screenshot shows the Postman interface for the same POST request. The 'Headers' tab is selected. A table titled 'Headers' shows one entry: 'token' with a checked checkbox under 'KEY'. The 'Value' column contains a long JWT token. There are also two empty rows for 'Key' and 'Value' with a 'Description' column.

	KEY	VALUE	DESCRIPTION	Bulk Edit	Presets
<input checked="" type="checkbox"/>	token	eyJraWQiOiIxIiwiYWxnIjoiUlMyNTYifQ.eyJp...			
	Key	Value	Description		

JSON WEB TOKENS (JWT)

Testing JWT (iii)

- Authenticate with JWT(ii):

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST' selected as the method, the URL 'http://localhost:8080/HelioWorld/demo/hello/testJWT', and a 'Send' button. Below the header are tabs for 'Params', 'Authorization' (which is selected and highlighted in green), 'Headers (9)', 'Body' (which is also highlighted in red), 'Pre-request Script', 'Tests', and 'Settings'. To the right of these tabs is a 'Cookies' section. Under the 'Body' tab, there are several radio buttons for file types: 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected and highlighted in orange), 'binary', and 'GraphQL'. Below this, there is a text input field containing the value 'Guadalupe'. At the bottom of the interface, there is a summary bar with 'Body', 'Cookies', 'Headers (5)', and 'Test Results' tabs. The 'Test Results' tab is selected and highlighted in red. It shows a status message '200 OK 59 ms 165 B' and a 'Save Response' button. Below the summary bar, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'Text' (which is selected and highlighted in red). The main body area displays the response 'Hello Guadalupe'.

Contents

1. API Keys
2. JWT
3. JWE

JSON WEB ENCRYPTION (JWE)

Introduction

- JSON Web Encryption:
- It is encrypted using cryptographic algorithms and serialized for tokenization in HTTP authorization headers.
- In order to ensure the message or token has not been altered in any way the token contains a digital signature (JWS) that is cryptographically encrypted using a strong algorithm such as HMAC SHA-256.

JSON WEB ENCRYPTION (JWE)

Communication Flow

1. Initial request with username and password
2. Response with 200 status code and encrypted JSON Web Token
3. Following requests include the JWE in the header
4. Responses with 200 status code and response

JSON WEB ENCRYPTION (JWE)

Implementation

- Create required static variables
- Create the method to obtain the token
- Create the method to be invoked with the token
- Pay attention to the new imports from *org.jose.jwe*

JSON WEB ENCRYPTION (JWE)

Create the encrypted key

```
static JsonWebKey jwKey = null;

static {
    // Setting up Direct Symmetric Encryption and Decryption
    String jwkJson = "{\"kty\":\"oct\",\"k\":\"9d6722d6-b45c-4dcb-bd73-2e057c44eb93-928390\"}";
    try {
        new JsonWebKey.Factory();
        jwKey = Factory.newJwk(jwkJson);
    } catch (JoseException e) {
        e.printStackTrace();
    }
}
```

JSON WEB ENCRYPTION (JWE)

Authenticate credentials (i)

```
@Path("/authenticateJWE")
@POST
@Produces(MediaType.APPLICATION_JSON)
public Response authenticateCredentialsJWE(
    @HeaderParam("username") String username,
    @HeaderParam("password") String password)
    throws JsonGenerationException, JsonMappingException,
    IOException {

    User user = new User();
    user.setUser(username);
    user.setPassword(password);
```

```
JwtClaims claims = new JwtClaims();
claims.setIssuer("uca");
claims.setExpirationTimeMinutesInTheFuture(10);
claims.setGeneratedJwtId();
claims.setIssuedAtToNow();
claims.setNotBeforeMinutesInThePast(2);
claims.setSubject(user.getUser());
claims.setStringListClaim("roles", "admin");

JsonWebSignature jws = new JsonWebSignature();
```

JSON WEB ENCRYPTION (JWE)

Authenticate credentials (ii)

```
jws.setPayload(claims.toJson());
jws.setKeyIdHeaderValue(jwKey.getKeyId());
jws.setKey(jwKey.getKey());
jws.setAlgorithmHeaderValue(AlgorithmIdentifiers.HMAC_SHA256);
```

```
String jwt = null;
try {
    jwt = jws.getCompactSerialization();
} catch (JoseException e) {    e.printStackTrace(); }
```

```
JsonWebEncryption jwe = new JsonWebEncryption();
```

JSON WEB ENCRYPTION (JWE)

Authenticate credentials (iii)

```
jwe.setAlgorithmHeaderValue(  
    KeyManagementAlgorithmIdentifiers.DIRECT);  
  
jwe.setEncryptionMethodHeaderParameter(  
  
ContentEncryptionAlgorithmIdentifiers.AES_128_CBC  
_HMAC_SHA_256);  
  
jwe.setKey(jwKey.getKey());  
  
jwe.setKeyIdHeaderValue(jwKey.getKeyId());  
  
jwe.setContentTypeHeaderValue("JWT");  
  
jwe.setPayload(jwt);  
  
String jweSerialization = null;  
try {  
    jweSerialization = jwe.getCompactSerialization();  
} catch (JoseException e) {    e.printStackTrace(); }  
  
return Response.status(200).entity(jweSerialization).build();  
}
```

JSON WEB ENCRYPTION (JWE)

Test JWE (i)

```
@POST  
@Path("/testJWE")  
@Consumes(MediaType.TEXT_PLAIN)  
@Produces(MediaType.TEXT_PLAIN)  
  
public Response testJWE (@HeaderParam("token")  
String token, String myName)  
  
throws JsonGenerationException,  
JsonMappingException, IOException {
```

```
JwtConsumer jwtConsumer = new  
JwtConsumerBuilder()  
  
.setRequireExpirationTime()  
  
.setAllowedClockSkewInSeconds(30)  
  
.setRequireSubject()  
  
.setExpectedIssuer("uca")  
  
.setDecryptionKey(jwKey.getKey())  
  
.setVerificationKey(jwKey.getKey()).build();
```

JSON WEB ENCRYPTION (JWE)

Test JWE (ii)

```
try {  
    // Validate the JWT and process it to the Claims  
    JwtClaims jwtClaims = jwtConsumer.processToClaims(token);  
    System.out.println("JWE validation succeeded! " + jwtClaims);  
} catch (InvalidJwtException e) {  
    System.out.println("JWE is Invalid: " + e);  
    return Response.status(Status.FORBIDDEN.getStatusCode()).entity("Forbidden").build();  
}  
  
String sayHello="Hello "+myName;  
return Response.status(200).entity(sayHello).build();  
}
```

JSON WEB ENCRYPTION (JWE)

Testing JWE (i)

- Get authentication credentials (i)

The screenshot shows the Postman application interface. At the top, it displays a POST method, the URL <http://localhost:8080/HelWorld/demo/hello/authenticateJWE>, and a blue 'Send' button. Below the URL, there are tabs for Params, Auth (which is selected), Headers (9), Body, Pre-req., Tests, and Settings. A red 'Cookies' link is also visible. On the left, under the 'Auth' tab, the 'Type' dropdown is set to 'Digest Auth'. A note states: 'The authorization header will be automatically generated when you send the request.' Below this is another note: 'By default, Postman will extract values from the received response, add it to'. In the main area, there are fields for 'Username' containing 'restUser2' and 'Password' containing a series of dots ('.....'). A checkbox labeled 'Show Password' is unchecked. A warning message in a box says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' with a link 'Learn more about variables'.

JSON WEB ENCRYPTION (JWE)

Testing JWE (ii)

- Get authentication credentials (ii)
- Copy the obtained token

The screenshot shows a POST request in Postman to the URL `http://localhost:8080/HeloWorld/demo/hello/authenticateJWE`. The 'Headers' tab is selected, showing two fields: 'username' and 'password', both with checked checkboxes. The 'Body' tab is selected, showing a JSON response with a single key-value pair: 'Value' (containing a long base64 string).

KEY	VALUE	DESCRIP	Bulk Edit	Presets
username	username	ooo		
password	password			

Key	Value	Description

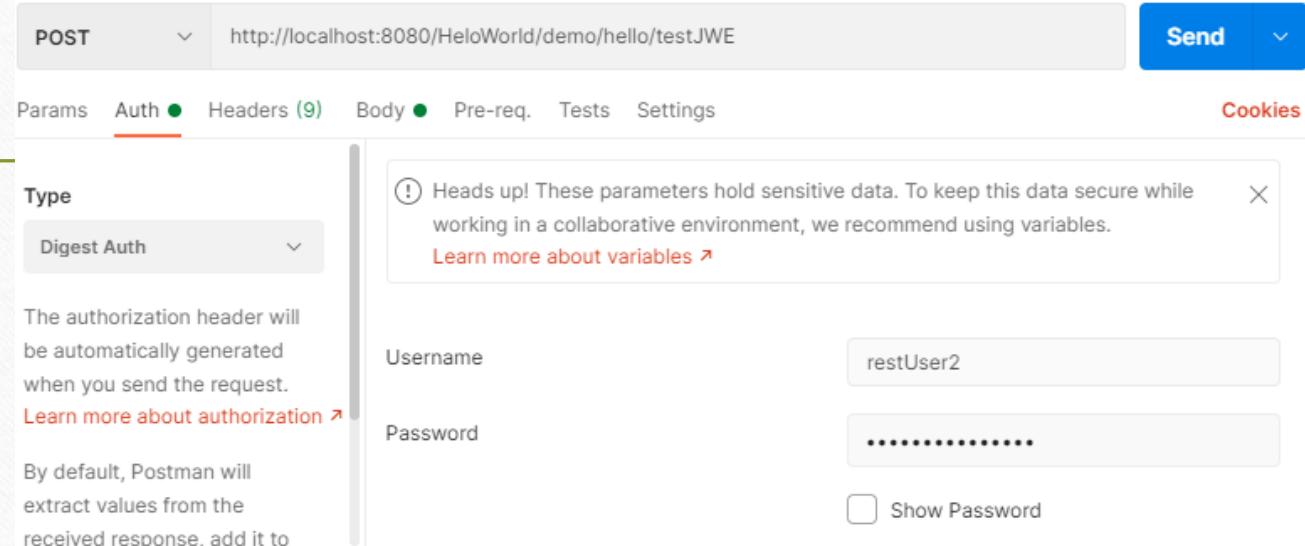
Body Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON ↻ 200 OK 29 ms 625 B Save Response

```
1 eyJhbGciOiJkaXiiLCJ1bmMiOiJBMTI4Q0JDLUhTMjU2Iiwia2lkIjpuWxsLCJjdHkiOiJKV1QiifQ..nOKDPLPMPEiAbVZ1NBEBmA.-UrObMsPloTRwnxskCx_EeAMgw3ugXHBfF04bTan5C88Xbxe7wxCSkeKy_VnZxW0V45XjVu8Pz0Y1AN3DGSKq608ttJkctCvSTIw6eBbUKJYqlze4HIR8VWoSftEF7_dFTUL4Q6up9_Loje52aBiwtSiPiolABIftBDhd0UsuTGK8_wa0qXaMh0uVF0JGdzGt8PJ4kwSVmr-cj4N-FP5WlrA6MJ5hNpPRH6S8puBSh6wxunKp5e4xngjsRINoxqMZu4-V9-18yU-DDG100zaEGm9st4xvLgN1mHRHQNT0sciG03r0Jh_9HTGGR7szVKjv4a5Z-VCKk8X2YXoIJ1FA.GqQc_dI-88bojlniSi9CQA
```

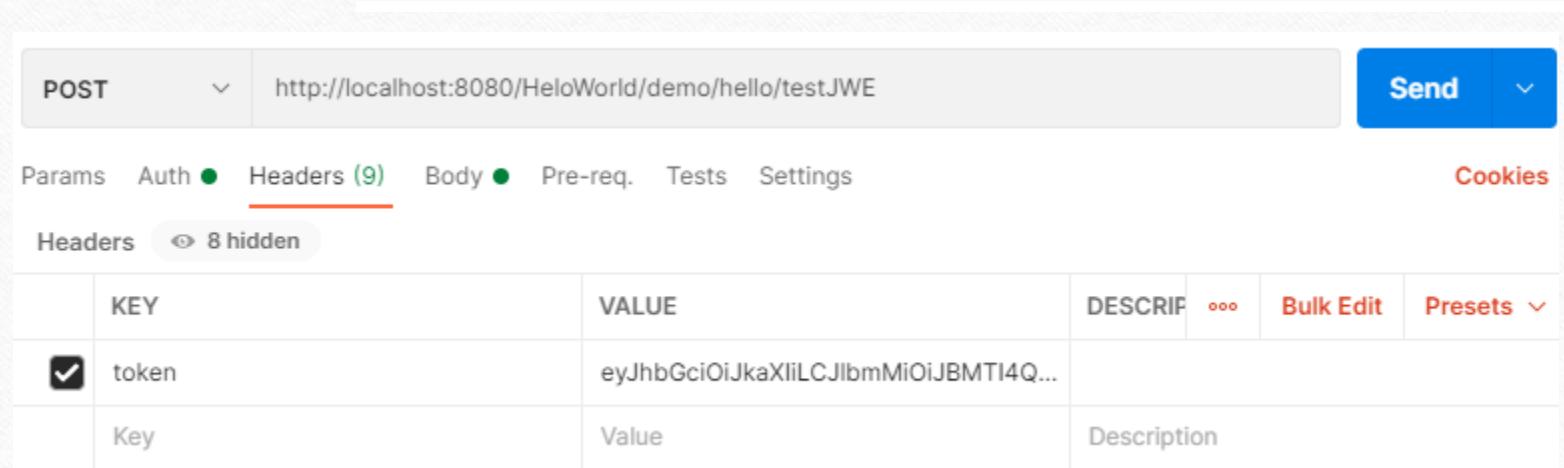
JSON WEB ENCRYPTION (JWE)

Testing JWE (iii)

- Authenticate with JWE(i):
- Paste the obtained token in a new header param



The screenshot shows the Postman interface for a POST request to `http://localhost:8080/HelоАWorld/demo/hello/testJWE`. The 'Auth' tab is selected. The 'Type' dropdown is set to 'Digest Auth'. A note on the right says: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' It includes links to 'Learn more about variables' and 'Learn more about authorization'. The 'Username' field contains 'restUser2' and the 'Password' field contains a redacted password. A 'Show Password' checkbox is unchecked.



The screenshot shows the Postman interface for the same POST request. The 'Headers' tab is selected. There are 9 headers defined. One header, 'token', is checked and has a value of `eyJhbGciOiJkaXliLCJlbmMiOiJBMTI4Q...`. Other headers listed are 'Key' and 'Value'.

KEY	VALUE	DESCRIP	...	Bulk Edit	Presets
token	eyJhbGciOiJkaXliLCJlbmMiOiJBMTI4Q...		ooo		
Key	Value	Description			

JSON WEB ENCRYPTION (JWE)

Testing JWE (iv)

- Authenticate with JWE(ii):

The screenshot shows a POST request in the Postman application. The URL is `http://localhost:8080/HeloWorld/demo/hello/testJWE`. The 'Body' tab is selected, showing the input field contains the value `Guadalupe`. The response at the bottom shows a 200 OK status with the message `Hello Guadalupe`.

POST http://localhost:8080/HeloWorld/demo/hello/testJWE

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw Text

1 Guadalupe

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

1 Hello Guadalupe

200 OK 14 ms 165 B Save Response

Support Bibliography and References

- Amauri Valdés, Developers Corner. <https://avaldes.com/category/java-development/jax-rs/>
- Andrés Salazar C., René Enríquez. RESTFUL Java Web Services Security.
<https://www.safaribooksonline.com/library/view/restful-java-web/9781783980109/>
- Bill Burke. RESTFUL Java with JAX-RS 2.0.
<https://www.safaribooksonline.com/library/view/restful-java-with/9781449361433/>
- OWASP. REST Security Cheat Sheet.
https://www.owasp.org/index.php/REST_Security_Cheat_Sheet#Introduction
- Guy Levin. Top 5 REST API Security Guidelines. <https://dzone.com/articles/top-5-rest-api-security-guidelines>
- Brian Campbell. Jose4j/JWT Examples.
https://bitbucket.org/b_c/jose4j/wiki/JWT%20Examples