DEPARTMENT OF COMPUTER SCIENCE

VEER NARMAD SOUTH GUJARAT UNIVERSITY, SURAT

PROJECT REPORT

AS A PARTIAL REQUIREMENT

FOR THE DEGREE OF

MASTER OF COMPUTER APPLICATION

(M.C.A /6TH SEMESTER)

YEAR: 2020-21

JSON Web Token Authentication

GUIDED BY:

Mr. Rohit Gaur

SUBMITTED BY:

Harshita Bagaria

ORGANIZATION

Department of Computer Science, VNSGU

SURAT

Department of Computer Science

Veer Narmad South Gujarat University

Udhna Magdalla Road, Vesu, Surat – 395007 (Gujarat) INDIA

CERTIFICATE

This is to certify that the seminar entitled ______________

JSON Web Token Authentication

has been carried out by Mr. / Ms. Harshita Bagaria

____

of M.C.A Semester VI Exam No. 3 *as a partial fulfillment*

*of the course, for the Academic Year 20*20 *- 20*21

Date:

Internal Guide Name & Sign

SEMINAR OF MCA
Academic Year ___________
Approved by: ______________________
____
____
(Examiners)

Professor & Head
Dept. of Computer Science

ACKNOWLEDGEMENT

In the present world of competition there is a race of existence in which those are having will come forward succeed. Project is like a bridge between theoretical and practical working. First of all, I would like to thank the supreme power the almighty God who is obviously the one who has always guided me to work on the right path of life. Without his grace this project could not become a reality. Next to him are my parents, whom I am greatly indebted for me to brought up with love and encouragement to this stage. I thank to **Mr. Rohit Gaur** , for providing me an opportunity to do the seminar and giving us all support and guidance, which made me complete the project duly. I am extremely thankful for providing such a nice support and guidance, although he had busy schedule. I would like to express my deep sense of gratitude towards **Department of Computer Science, VNSGU, Surat.**

Providing me this opportunity to implement the theoretical knowledge into practical work as a part of seminar in Sixth semester curriculum. I am feeling oblige in taking the opportunity to sincerely thanks my worthy teacher of Computer Application Mr. Rohit Gaur.

At last I am thankful to all my teacher, friends and my colleagues who have been always helping and encouraging me throughout the year .

INDEX

What is JSON?

JSON stands for **J**ava**S**cript **O**bject **N**otation.

JSON is a text format for storing and transporting data.

JSON is “self-describing” and easy to understand.

JSON is a lightweight data-interchange format. It is plain text written in JavaScript object Notation. It is used to send data between computers. It is language independent. Code for reading and generating JSON exists in many programming languages.

The JSON format was originally specified by Douglas Crockford.

Why use JSON?

The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

JSON.parse() -

JavaScript has a built-in function for converting JSON strings into JavaScript objects.

JSON.stringify() -

JavaScript also has a built in function for converting an object into a JSON string.

You can receive pure text from a server and use it as a JavaScript object.

You can send a JavaScript object to a server in pure text format.

You can work with data as JavaScript objects, with no complicated parsing and translations.

Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

JSON WEB TOKEN AUTHENTICATION

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/ value pairs
- Data is sepearted by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON Data – A Name and a Value

JSON data is written as name/ value pairs.

A name/ value pair consists of a filed name(in double quotes), followed by a colon, followed by a value:

Example

“name”:”John”

Note : JSON name requires double quotes

JSON-Evaluates to JavaScript Objects

The JSON format is almost identical to JavaScript objects.

In JSON, *keys* must be strings, written with double quotes:

Example

JSON- {“name”:”John”}

In JavaScript, keys can be strings, numbers, or identifier names:

Example

JavaScript – {name:”John”}

JSON Values

In **JSON**, *values* must be one of the following data types:

- a string
- a number
- an object
- an array
- a boolean
- null

In **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

- a function
- a date
- undefined

In JSON, *string values* must be written with double quotes:

JavaScript Objects

JSON syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an object and assign data to it, like this:

Example:

```
person = {name:"John", age:31, city:"New York"};
```

You can access a JavaScript object like this:

Example:

```
person.name;
```

It can also be accessed like this:

Example:

```
person["name"];
```

Data can be modified like this:

Example:

```
person.name = "Gilbert";
```

It can also be modified like this:

Example:

```
person["name"] = "Gilbert";
```

JavaScript Arrays as JSON

The same way JavaScript objects can be written as JSON, JavaScript arrays can also be written as JSON.

JSON Files

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

What is JSON Web Token?

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on *signed* tokens. Signed tokens can verify the *integrity* of the claims contained within it, while encrypted tokens *hide* those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

When should you use JSON Web?

Here are some scenarios where JSON Web Tokens are useful:

- **Authorization**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

- **Information Exchange**: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

What is JSON Web Token structure?

JSON Web Tokens consist of three parts separated by dots (.), which are

- : Header
- Payload
- Signature

JWT typically looks like the following-

xxxxxx,yyyyy,zzzzz

Header

The header *typically* consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

Example :

```
{
        “alg” : “HS256”,
        “typ” : “JWT”
}
```

Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

Payload

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: *registered*, *public*, and *private* claims.

- **Registered claims:** These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims.

Some of them are: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), and others.

- **Public Claims :** These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.

- **Private Claims :** These are the custom claims created to share information between parties that agree on using them and are neither *registered* or *public* claims.

  Example :

  ```
  {
          "sub" : "1234567890",
          "name" : "John Doe"
          "admin" : true
  }
  ```

The payload is then **Base64Url** encoded to form the second part of the JSON Web Token.

Signature

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

Example : if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:

```
HMACSHA256(
        base64UrlEncode(header) + "." +
```

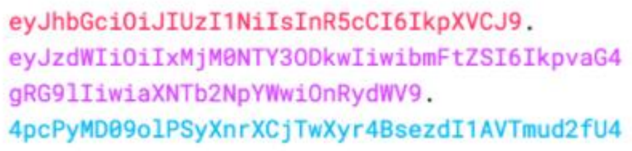
```
base64UrlEncode(payload),
secret)
```

The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

Putting all together

The output is three Base64-URL strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML.

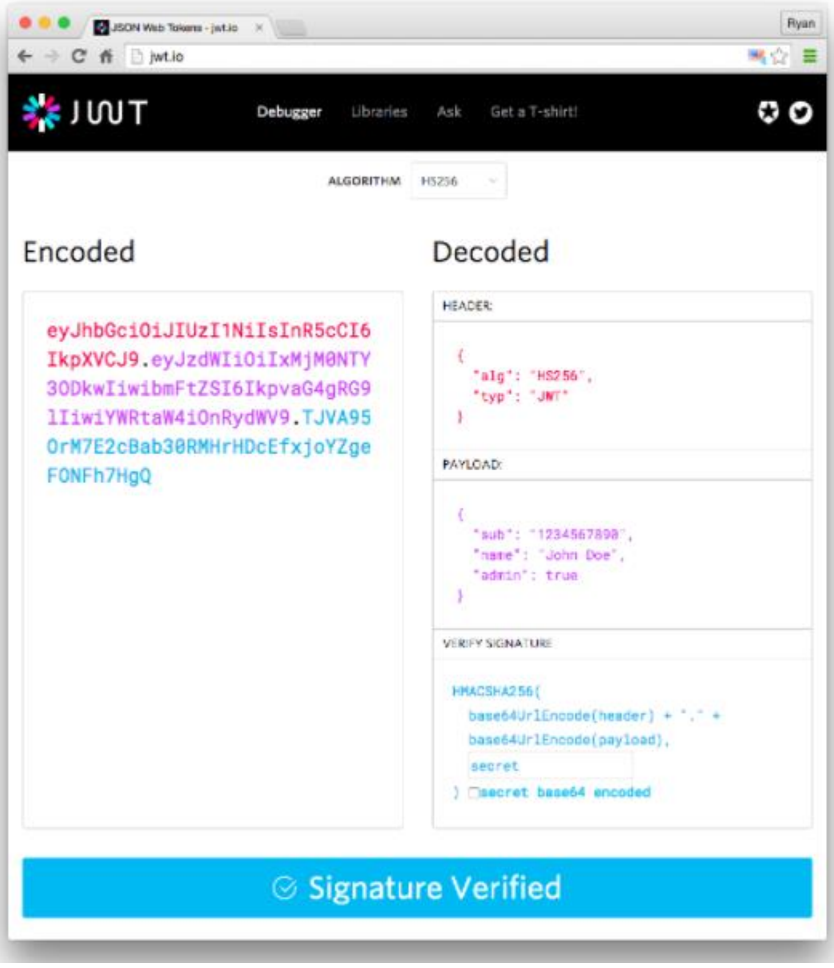
The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

If you want to play with JWT and put these concepts into practice, you can use [jwt.io Debugger]() to decode, verify, and generate JWTs.

How do JSON Web Token work?



ALGORITHM HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxjoYZge
FONFh7HgQ
```

Decoded

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned. Since tokens are credentials, great care must be taken to prevent security issues. In general, you should not keep tokens longer than required.

You also should not store sensitive session data in browser storage due to lack of security.

Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the **Authorization** header using the **Bearer** schema. The content of the header should look like the following:

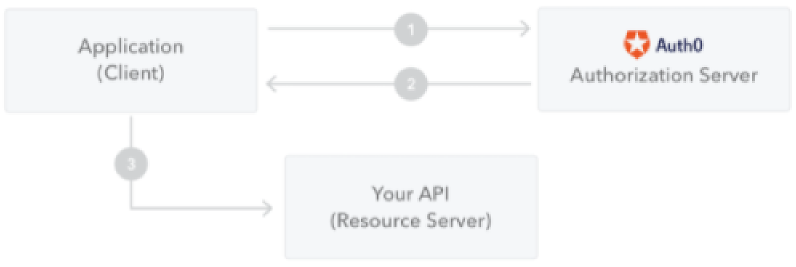
JSON WEB TOKEN AUTHENTICATION

Authorization: Bearer <token>

This can be, in certain cases, a stateless authorization mechanism. The server's protected routes will check for a valid JWT in the Authorization header, and if it's present, the user will be allowed to access protected resources. If the JWT contains the necessary data, the need to query the database for certain operations may be reduced, though this may not always be the case.

If the token is sent in the Authorization header, Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.

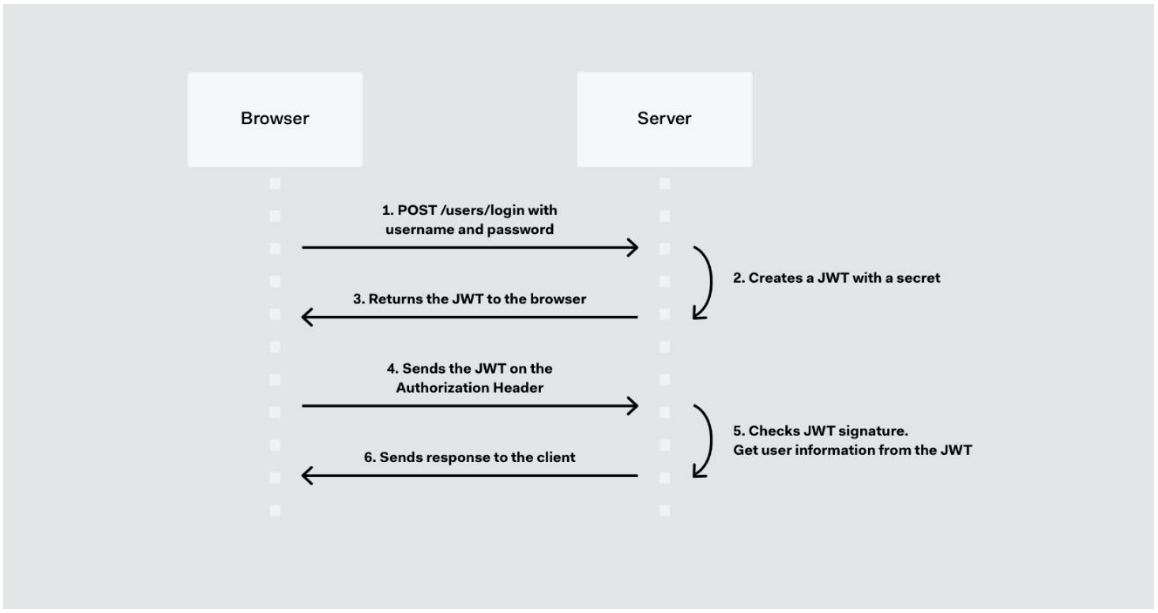
The following diagram shows how a JWT is obtained and used to access APIs or resources:



1. The application or client requests authorization to the authorization server. This is performed through one of the different authorization flows. For example, a typical OpenID Connect compliant web application will go through the /oauth/authorize endpoint using the authorization code flow.

2. When the authorization is granted, the authorization server returns an access token to the application.

3. The application uses the access token to access a protected resource (like an API).


Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the browser
4. Sends the JWT on the Authorization Header
5. Checks JWT signature. Get user information from the JWT
6. Sends response to the client

Why should we use JSON Web Token?

The benefits of **JSON Web Tokens (JWT)** when compared to **Simple Web Tokens (SWT)** and **Security Assertion Markup Language Tokens (SAML)**.

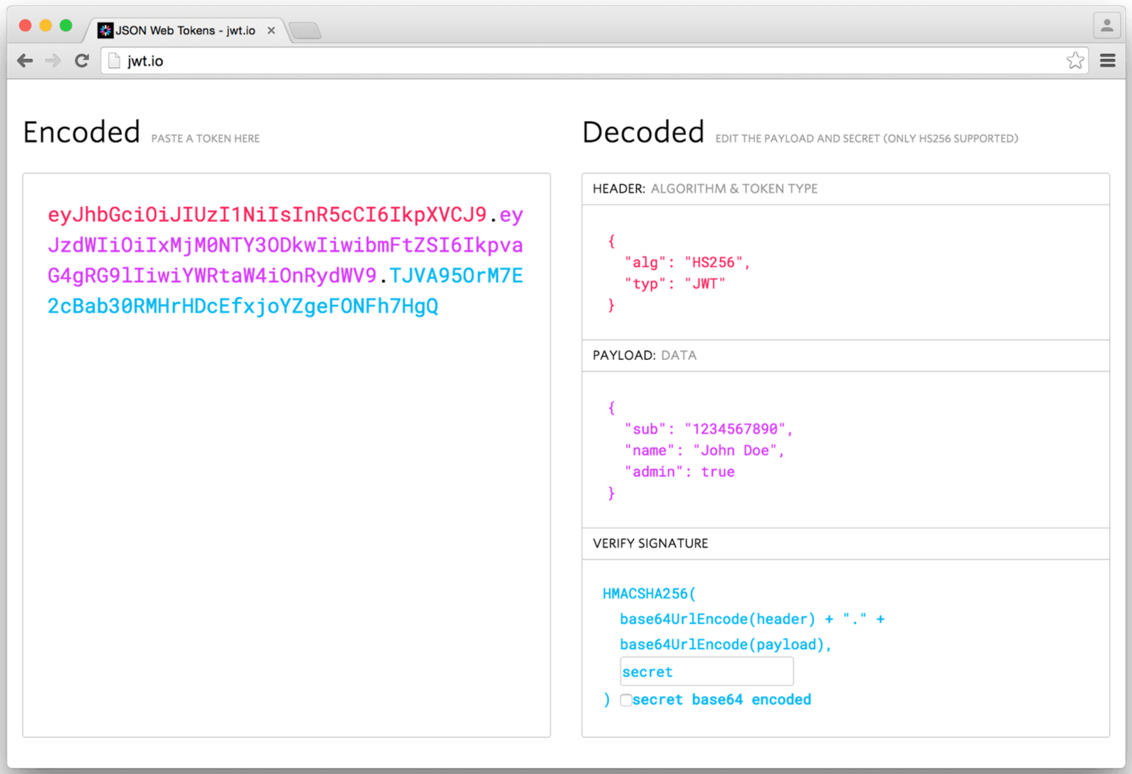
As JSON is less verbose than XML, when it is encoded its size is also smaller, making JWT more compact than SAML. This makes JWT a good choice to be passed in HTML and HTTP environments.

Security-wise, SWT can only be symmetrically signed by a shared secret using the HMAC algorithm. However, JWT and SAML tokens can use a public/private key pair in the form of a X.509 certificate for signing. Signing XML with XML Digital Signature without introducing obscure security holes is very difficult when compared to the simplicity of signing JSON.

JSON parsers are common in most programming languages because they map directly to objects. Conversely, XML doesn't have a natural document-to-object mapping. This makes it easier to work with JWT than SAML assertions.

Regarding usage, JWT is used at Internet scale. This highlights the ease of client-side processing of the JSON Web token on multiple platforms, especially mobile.

JSON WEB TOKEN AUTHENTICATION



Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

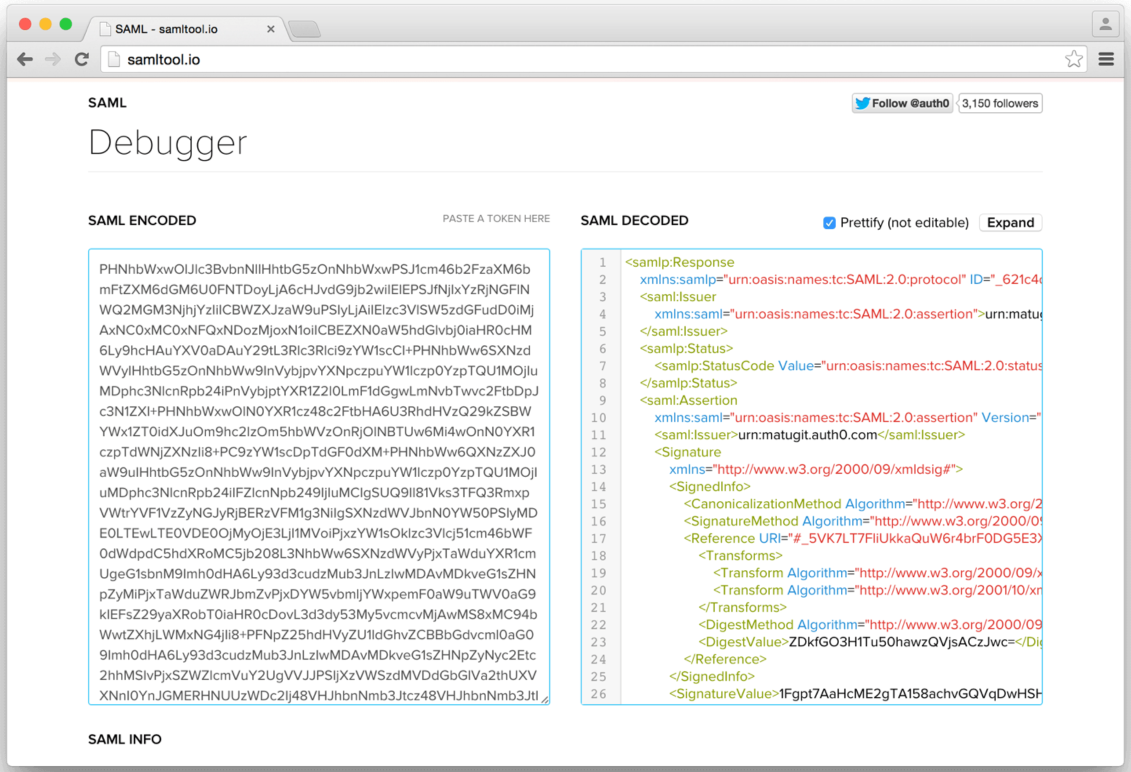
PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

VS



SAML

Debugger

SAML ENCODED PASTE A TOKEN HERE

PHNhbWxwOlJlc3BvbnNlIHhtbG5zOnNhbWxwPSJ1cm46b2FzaXM6b
mFtZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wiIElEPSJfNjIxYzRjNGFlN
WQ2MGM3NjhjYzliICBWZXJzaW9uPSIyLjAiIElzc3VlSW5zdGFudD0iMj
AxNC0xMC0xNFQxNDozMjoxN1oiICBEZXN0aW5hdGlvbj0iaHR0cHM
6Ly9hcHAuYXV0aDAuY29tL3Rlc3Rlci9zYW1scCI+PHNhbWw6SXNzd
WVyIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIu
MDphc3NlcnRpb24iPnVybjptYXR1Z2l0LmF1dGgwLmNvbTwvc2FtbDpJ
c3N1ZXI+PHNhbWxwOlN0YXR1cz48c2FtbHA6U3RhdHVzQ29kZSBW
YWx1ZT0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOnN0YXR1
czpTdWNjZXNzIi8+PC9zYW1scDpTdGF0dXM+PHNhbWw6QXNzZXJ0
aW9uIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjI
uMDphc3NlcnRpb24iIFZlcnNpb249IjIuMCIgSUQ9Il81Vks3TFQ3Rmxp
VWtrYVF1VzZyNGJyRjBERzVFM1g3NiIgSXNzdWVJbnN0YW50PSIyMD
E0LTEwLTE0VDE0OjMyOjE3LjI1MVoiPjxzYW1sOklzc3Vlcj51cm46bWF
0dWdpdC5hdXRoMC5jb208L3NhbWw6SXNzdWVyPjxTaWduYXR1cm
UgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHN
pZyMiPjxTaWduZWRJbmZvPjxDYW5vbmljYWxpemF0aW9uTWV0aG9
kIEFsZ29yaXRobT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8xMC94b
WwtZXhjLWMxNG4jIi8+PFNpZ25hdHVyZU1ldGhvZCBBbGdvcml0aG0
9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHNpZyNyc2Etc
2hhMSIvPjxSZWZlcmVuY2UgVVJJPSIjXzVWSzdMVDdGbGlVa2thUXV
XNnI0YnJGMERHNUUzWDc2Ij48VHJhbnNmb3Jtcz48VHJhbnNmb3JtI

SAML DECODED ☑ Prettify (not editable) Expand

```
<samlp:Response
   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" ID="_621c4
   <saml:Issuer
      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:matu
   </saml:Issuer>
   <samlp:Status>
      <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status
   </samlp:Status>
   <saml:Assertion
      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="
      <saml:Issuer>urn:matugit.auth0.com</saml:Issuer>
      <Signature
         xmlns="http://www.w3.org/2000/09/xmldsig#">
         <SignedInfo>
            <CanonicalizationMethod Algorithm="http://www.w3.org/2
            <SignatureMethod Algorithm="http://www.w3.org/2000/0
            <Reference URI="#_5VK7LT7FliUkkaQuW6r4brF0DG5E3
               <Transforms>
                  <Transform Algorithm="http://www.w3.org/2000/09/
                  <Transform Algorithm="http://www.w3.org/2001/10/x
               </Transforms>
               <DigestMethod Algorithm="http://www.w3.org/2000/09
               <DigestValue>ZDkfGO3H1Tu50hawzQVjsACzJwc=</Di
            </Reference>
         </SignedInfo>
         <SignatureValue>1Fgpt7AaHcME2gTA158achvGQVqDwHSH
```

SAML INFO

How we use JSON Web Tokens in AuthO?

In Auth0, we issue JWTs as a result of the authentication process. When the user logs in using Auth0, a JWT is created, signed, and sent to the user. Auth0 supports signing JWT with both HMAC and RSA algorithms. This token will be then used to authenticate and authorize with APIs which will grant access to their protected routes and resources.

We also use JWTs to perform authentication and authorization in Auth0's API v2, replacing the traditional usage of regular opaque API keys. Regarding authorization, JSON Web Tokens allow granular security, that is the ability to specify a particular set of permissions in the token, which improves debuggability.

Advantages

- **No Session to Manage (stateless):** The JWT is a self contained token which has authetication information, expire time information, and other user defined claims digitally signed.

- **Portable:** A single token can be used with multiple backends.

- No Cookies Required, So It's Very Mobile Friendly.

- **Good Performance:** It reduces the network round trip time.

- **Decoupled/Decentralized:** The token can be generated anywhere. Authentication can happen on

  the resource server, or easily seperated into its own server.

Disadvantages

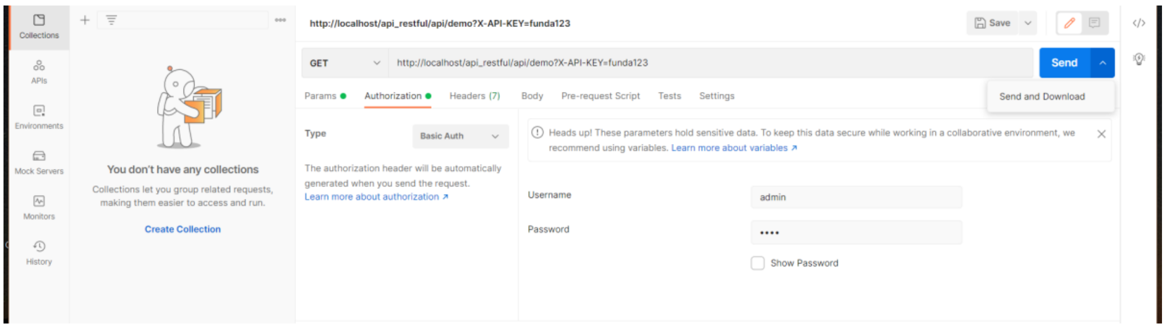
- **Compromised Secret Key :** The best and the worst thing about JWT is that it relies on just one *Key*. Consider that the ***Key* is leaked by a careless or a rogue** developer/administrator, the whole system is compromised!

- **Cannot push Messages to clients** (Identifying clients from server) **:** As we have no record about the logged-in clients on the DB end, we cannot push messages to all the clients.

- **Crypto-algo can be deprecated**: JWT relies completely on the Signing algorithm. Now, though it is not frequent, but in the past many Encryption/Signing algorithms have been deprecated.

- **Data Overhead :** The size of the JWT token will be more than that of a normal Session token. The more data you add in the JWT token, the longer it gets linearly. Remember, each request needs the token in it for request verification. So say, a 1 KB JWT token implies each request will have 1KB over-head upload which is really bad in cases of low speed net connectivity.

Screenshots



http://localhost/api_restful/api/demo?X-API-KEY=funda123

GET http://localhost/api_restful/api/demo?X-API-KEY=funda123

Params ● Authorization ● Headers (7) Body Pre-request Script Tests Settings

You don't have any collections

Collections let you group related requests, making them easier to access and run.

Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn more about authorization ↗

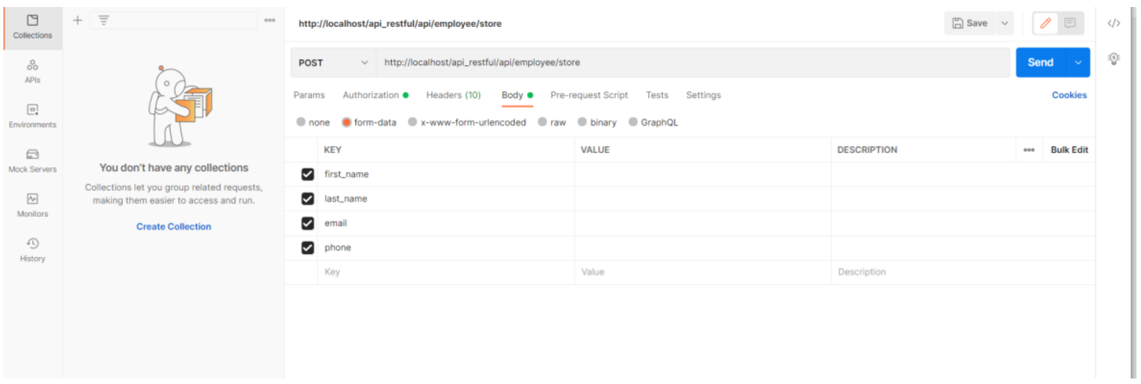
Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Username: admin

Password: ••••

Show Password

First we have to login with valid credentials.
If invalid credentials then you will not be able to login.



You don't have any collections

Collections let you group related requests, making them easier to access and run.

http://localhost/api_restful/api/employee/store

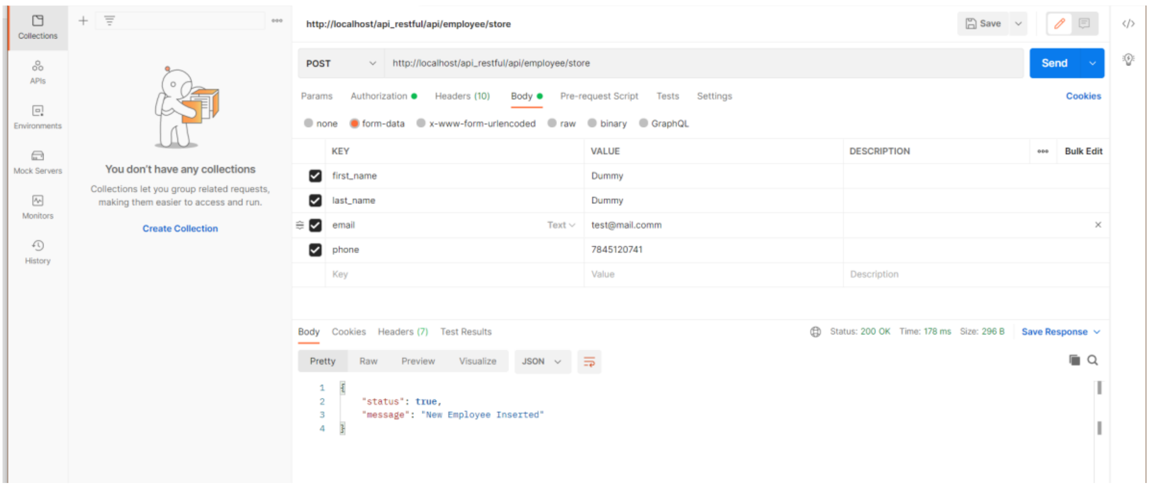
POST http://localhost/api_restful/api/employee/store

Params | Authorization ● | Headers (10) | Body ● | Pre-request Script | Tests | Settings

none | ● form-data | x-www-form-urlencoded | raw | binary | GraphQL

	KEY	VALUE	DESCRIPTION
✓	first_name		
✓	last_name		
✓	email		
✓	phone		
	Key	Value	Description

After successful login you will be enter the details in the field.



http://localhost/api_restful/api/employee/store

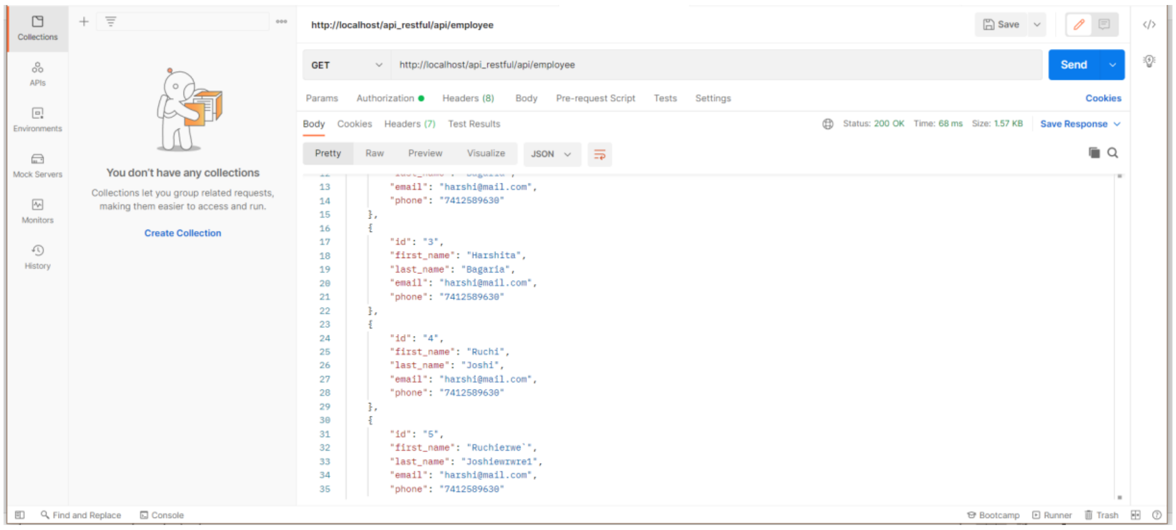
POST http://localhost/api_restful/api/employee/store

	KEY	VALUE	DESCRIPTION
✓	first_name	Dummy	
✓	last_name	Dummy	
✓	email	test@mail.comm	
✓	phone	7845120741	
	Key	Value	Description

Status: 200 OK Time: 178 ms Size: 296 B

```
{
    "status": true,
    "message": "New Employee Inserted"
}
```

After filling up the data’s you will get the message for successful inserting.



http://localhost/api_restful/api/employee

GET http://localhost/api_restful/api/employee

Status: 200 OK Time: 68 ms Size: 1.57 KB

```
        "email": "harshi@mail.com",
        "phone": "7412589630"
    },
    {
        "id": "3",
        "first_name": "Harshita",
        "last_name": "Bagaria",
        "email": "harshi@mail.com",
        "phone": "7412589630"
    },
    {
        "id": "4",
        "first_name": "Ruchi",
        "last_name": "Joshi",
        "email": "harshi@mail.com",
        "phone": "7412589630"
    },
    {
        "id": "5",
        "first_name": "Ruchierwe`",
        "last_name": "Joshiewrwre1",
        "email": "harshi@mail.com",
        "phone": "7412589630"
```

Here, you can see all the inserted data in the table.

Bibliography

- https://jwt.io/introduction