INFO3180 - LECTURE 10

HTTP HEADERS, RESTFUL API'S AND JSON WEB TOKENS (JWT)

RESTFUL API'S

What is an API?

It stands for Application Programming Interface.

An API is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact.

You can also think of a Web API/Service as a way of exposing your data to different users or applications.

What is REST?

It stands for REpresentational State Transfer

It has become the standard architectural design for Web Services/API's.

REST CONSTRAINTS

- ***Client-Server***
- ***Stateless***
- ***Cacheable***
- ***Layered System***
- ***Uniform Interface***
- ***Code on Demand (Optional)***

It usually uses the HTTP protocol

COMMON HTTP METHODS

- ***GET: Retrieve data from a specific resource***
- ***POST: Submit data to be processed to a specific resource***
- ***PUT: Update a specific resource***
- ***DELETE: Deletes a specific resource***

Many developers will try to write their Web API's like this...

HTTP Verb	Path	Used For
GET	/api/listTasks	Get list of tasks
GET	/api/showTask/1	Get a single task
GET	/api/getCompleted	Get completed tasks
POST	/api/createTasks	Creates a new task
POST	/api/updateTask/1	Updates a task
POST	/api/tasks/1	Deletes a task

But instead you should try to write your RESTful Web API's like this...

HTTP Verb	Path	Used For
GET	/api/tasks	Get list of tasks
GET	/api/tasks/1	Get a single task
POST	/api/tasks	Creates a new task
PATCH/PUT	/api/tasks/1	Updates a task
DELETE	/api/tasks/1	Deletes a task

Usually for RESTful API's our endpoints will return/output some data in either JSON or XML format.

The preference for many API's these days is to use JSON. But how do we do this in Flask?

We can use Flask's built in `jsonify` method to help us output our data in JSON format.

```python
from flask import jsonify

@app.route("/api/tasks")
def tasks():
    tasks = [{'id': 1, 'title': 'Hello'},
{'id': 2, 'title': 'World'}]

    return jsonify(tasks=tasks)
```

Here we use Flask's `jsonify` method to output the data as in JSON format.

JSON OUTPUT FROM OUR ROUTE

```
{
  "tasks": [
    {
      "id": 1,
      "title": "Hello"
    },
    {
      "id": 2,
      "title": "World"
    }
  ]
}
```

It is also important that we return the proper HTTP Status Codes.

COMMON HTTP STATUS CODES

- *200 OK*
- *201 Created*
- *202 Accepted*
- *400 Bad Request*
- *401 Unauthorized*
- *403 Forbidden*
- *404 Not Found*
- *500 Internal Server Error*

If you wanted to quickly set the status code for your API response then you can do it by add the status code after the jsonify() function separated by a comma. (e.g. , 201)

```
from flask import jsonify

@app.route("/api/tasks", methods=["POST"])
def add_task():
    # ...your code to create a new task
    message = "Task created successfully"

    return jsonify(message=message), 201
```

SOME SUGGESTED BEST PRACTICES FOR RESTFUL API'S

- ***Nouns are good; verbs are bad for naming resources***
- ***Keep it simple***
- ***Plural nouns and concrete names***
- ***Simplify associations e.g. `/users/620099999/tasks`***
- ***Use appropriate HTTP Status Codes***
- ***Use SSL (HTTPS)***
- ***Version your API (e.g. /api/v1/users)***

these are just a few.

HTTP HEADERS

HTTP headers allow the client and the server to pass additional information with the request or the response.

Source: Mozilla Developer Network – HTTP Headers

HTTP headers consist of name/value pairs separated by a colon ":".

```
NAME: Value
```

EXAMPLE HTTP REQUEST AND RESPONSE HEADERS

Name

- 0.0.0.0
- angular.js
- angular-route.js
- app.js
- tasks
- main.html

6 requests | 3.6KB transferred | Finish: 123…

× Headers Preview Response Cookies Timing

▶ **General**

▼ **Response Headers** view source

Cache-Control: public, max-age=0
Content-Length: 1596
Content-Type: text/html; charset=utf-8
Date: Mon, 20 Mar 2017 00:35:18 GMT
Server: Werkzeug/0.12.1 Python/2.7.13
X-UA-Compatible: IE=Edge,chrome=1

▼ **Request Headers** view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
Cookie: session=eyJfZnJlc2giOmZhbHNlLCJjc3JmX3Rva2VuIjp7IiBiIjoiWmpFeE5EZGxPRGszWW1OalptSXlaRFpoTWp
GF5MftFeXj54u_VY
Host: 0.0.0.0:8080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko)

How could we set a header in Flask or a JavaScript AJAX request?

In Plain JavaScript

```
var request = new XMLHttpRequest();
request.setRequestHeader("Content-Type","text/
plain");
```

Using the JavaScript fetch api

```
fetch('https://www.example.com/someapi',
{
  headers: {
    'content-type': 'application/json'
  }
});
```

*In Flask we can check for what headers were sent with a request by using the **request.headers** and **response.headers** objects.*

```
request.headers['Content-Type']
```

```
response.headers['X-Parachutes']
```

SECURING AN API

How do we secure an API?

One way is to use Token Authentication

TOKEN AUTHENTICATION

- ***A Client will use a token to make authenticated HTTP requests instead of a username and password.***
- ***Tokens can be set to expire after a period of time or can contain scope (access levels).***
- ***Tokens can be regenerated without needing to change a users password.***
- ***If a token is compromised, only their API access is generally affected.***
- ***We can then simply revoke the previous token and generate a new one.***

Authentication tokens are usually Base64encoded strings of data that are then passed as part of the request via the Authorization HTTP Header.

EXAMPLE AUTHORIZATION HEADER

```
Authorization: Basic
dGFza2x50nBhc3N3b3JkMTIzNA==
```

EXAMPLE AUTHORIZATION HEADER WITH AJAX REQUEST

```
fetch('/api/secure', {
    'headers': {
        // Try it with the `Basic` schema and you will see it
gives an error message.
        'Authorization': 'Basic dGFza2x50nBhc3N3b3JkMTIzNA=='
    }
})
    .then(function (response) {
        return response.json();
    })
    .then(function (response) {
        console.log(response)
    })
```

Which now leads us to another type of Authentication token called...

JSON WEB TOKENS (JWT)

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.

Source: JWT Introduction

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

Header.Payload.Signature

xxxxx.yyyyy.zzzzz

Source: JWT Introduction

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

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

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

Source: JWT Introduction

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional metadata.

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

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

Source: JWT Introduction

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.

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

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

Source: JWT Introduction

The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```


Source: JWT Introduction

EXAMPLE JWT AUTHORIZATION HEADER

```
Authorization: Bearer <token>
```

EXAMPLE JWT AUTHORIZATION HEADER WITH AJAX REQUEST

```
fetch('/api/secure', {
    'headers': {
        // JWT requires the Authorization schema to be `Bearer`
instead of `Basic`
        'Authorization': 'Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHr
HDcEfxjoYZgeFONFh7HgQ'
    }
})
    .then(function (response) {
        return response.json();
    })
    .then(function (response) {
        console.log(response)
    })
```

How can we generate JWT's in Python?

We can use the PyJWT library.

EXAMPLE USING PYJWT TO ENCODE AND DECODE A TOKEN

```
>>> import jwt
>>> encoded_jwt = jwt.encode({'some': 'payload'}, 'secret',
algorithm='HS256')

>>> encoded_jwt
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzb21lIjoicGF5bG9hZCJ9.4
twFt5NiznN84AWoo1d7KO1T_yoc0Z6XOpOVswacPZg'

// To decode the JWT and get back the payload
>>> jwt.decode(encoded_jwt, 'secret', algorithms=['HS256'])
{'some': 'payload'}
```

WHERE TO STORE YOUR JWTS

JWT tokens are typically stored in either:

- ***HTML5 Web Storage (localStorage or sessionStorage)***
- ***Cookies***

RESOURCES

- ***HTTP Headers - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers***
- ***Designing a RESTful API with Python and Flask - https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask***
- ***REST API Tutorial - http://www.restapitutorial.com/***
- ***JSON Web Tokens - https://jwt.io/***
- ***PyJWT - https://pyjwt.readthedocs.io/en/latest/***

RESOURCES

- ***Apiary - apiary.io/***
- ***Swagger - swagger.io***

DEMO