ENDPOINT

https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html

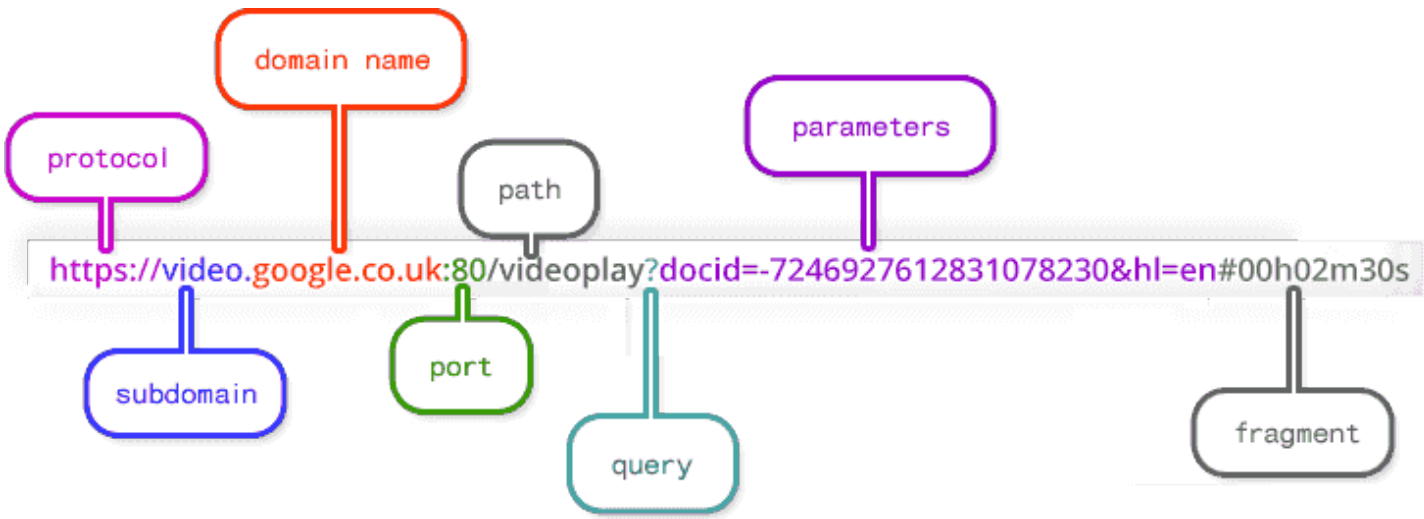
Parts of URL

Basic URL is made up of 3 parts:

- **Protocol** - declares how the browser should communicate with a web server. The most common is http (Hypertext Transfer Protocol) or https (Hypertext Transfer Protocol Secure)
- **Domain name** - unique reference that identifies a website on the internet
- **Path** - identifies the specific resource in the host that the web client wants to access

Complex URL might include:

- Protocol
- Subdomain
- Domain name
- Port
- Path
- Query
- Parameters
- Fragment



Query Parameters

They are extensions of the URL that are used to help define specific content or actions based on the data being passed. To append query params to the end of a URL, a ‘?’ is added followed immediately by a query parameter. To add multiple parameters, an ‘&’ is added in between each.

HTTP Headers

HTTP headers allow the client and the server to pass additional information with the request or the response. An HTTP header consists of its case-insensitive name followed by a colon ':', then by its value (without line breaks).

Request / Response

Web Endpoint Response Status
The default response status for an endpoint operation depends on the operation type (read, write, or delete) and what, if anything, the operation returns.

A **@ReadOperation** returns a value, the response status will be 200 (OK). If it does not return a value, the response status will be 404 (Not Found).

If a **@WriteOperation** or **@DeleteOperation** returns a value, the response status will be 200 (OK). If it does not return a value the response status will be 204 (No Content).

Consumes / Produces
Consumes
For a @WriteOperation (HTTP POST) that uses the request body, the consumes clause of the predicate is application/json. For all other operations the consumes clause is empty.

Produces
The produces clause of the predicate can be determined by the produces attribute of the @DeleteOperation, @ReadOperation, and @WriteOperation annotations. The attribute is optional. If it is not used, the produces clause is determined automatically.

If the operation method returns void or Void the produces clause is empty. If the operation method returns a Resource, the produces clause is application/octet-stream. For all other operations the produces clause is application/json.

```
@RestController
public class HeartbeatController {
    @GetMapping(value = "/heartbeat",
            produces = {MediaType.APPLICATION_JSON_VALUE})
```

```
    public ResponseEntity<?> getHeartbeat(@RequestHeader(value =
"userTokenAuth", required = false) String token) throws Exception {
        if (token == null || token.equals("")) {
            throw new GeneralException("Unauthorized",
HttpStatus.UNAUTHORIZED);
        }
        return new ResponseEntity<>(new SuccessfulQuery("Ok"),
HttpStatus.OK);
    }
}
```

Status Codes

Status codes are issued by a server in response to a client's request made to the server. The first digit of the status code specifies one of five standard classes of responses. The message phrases shown are typical, but any human-readable alternative may be provided.

- **1xx (Informational):** The request was received, continuing process
- **2xx (Successful):** The request was successfully received, understood, and accepted
- **3xx (Redirection):** Further action needs to be taken in order to complete the request
- **4xx (Client Error):** The request contains bad syntax or cannot be fulfilled
- **5xx (Server Error):** The server failed to fulfill an apparently valid request

Most common status codes
200 - OK
201 - Created
202 - Accepted

301- Moved Permanently
302 - Found
304 - Not Modified

400 - Bad Request
401 - Unauthorized
403 - Forbbiden
404 - Not Found

500 - Internal Server Error
503 - Service Unavailable

REST

Simply put, an endpoint is one end of a communication channel. When an API interacts with another system, the touchpoints of this communication are considered endpoints. For APIs, an endpoint can include a URL of a server or service. Each endpoint is the location from which APIs can access the resources they need to carry out their function.

REST APIs
A REST (REpresentational State Transfer) API allows the server to transfer to the client a representation of the state of the requested resource. It follows six constraints:

1. Uniform Interface
2. Client-server
3. Stateless
4. Cacheable
5. Layered System
6. Code on demand (optional)

My article about this: https://blog.usejournal.com/introduction-to-json-and-restful-apis-coding-bootcamp-series-7ad6aa294c89

HTTP / HTTPS

HTTPS (Hyper Text Transfer Protocol **Secure**) is the secure version of HTTP. In HTTPS, all the communications between the browser and the website are encrypted.

HTTPS pages typically use one of two secure protocols to encrypt communications - **SSL** (Secure Sockets Layer) or **TLS** (Transport Layer Security). Both the TLS and SSL protocols use what is known as an '**asymmetric' Public Key Infrastructure (PKI)** system. An asymmetric system uses **two 'keys'** to encrypt communications, a '**public**' key and a '**private**' key. Anything encrypted with the public key can only be decrypted by the private key and vice-versa. As the names suggest, the 'private' key should be kept strictly protected and should only be accessible the owner of the private key. In the case of a website, the private key remains securely enconced on the web server.

Examples:

- **Download pdf endpoint**

GET/POST/PUT * endpoint call or implementation or refactor
GET
The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

HEAD
The HEAD method asks for a response identical to that of a GET request, but without the response body.

POST

The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.

PUT
The PUT method replaces all current representations of the target resource with the request payload.

- **Modify status of an endpoint**
- **Add error handling to an endpoint**

```
 @GetMapping("/pitches")
  public ResponseEntity getPitches(@RequestHeader(value = "userTokenAuth",
required = false) String token) throws GeneralException {
    if (token != null && !token.equals("")) {
      return new ResponseEntity<>(pitchSetDTO, HttpStatus.OK);
    } else throw new GeneralException("Unauthorized",
HttpStatus.UNAUTHORIZED);
  }
```

```
public class GeneralException extends Exception {

  private ErrorMessage errorMessage;
  private HttpStatus httpStatus;

public GeneralException(String error, HttpStatus httpStatus) {
    this.errorMessage = new ErrorMessage(error);
    this.httpStatus = httpStatus;
  }
}
```

Expected skills (4/5)

- **Able to explain HTTP request communication flow**

In client-server protocols, like HTTP, sessions consist of three phases:

1. The client establishes a TCP connection (or the appropriate connection if the transport layer is not TCP).
2. The client sends its request, and waits for the answer.
3. The server processes the request, sending back its answer, providing a status code and appropriate data.

Sending a client request
Once the connection is established, the user-agent can send the request (a user-agent is typically a web browser, but can be anything else, a crawler, for example). A client request consists of text directives, separated by CRLF (carriage return, followed by line feed), divided into three blocks:

1. The first line contains a request method followed by its parameters:

- the path of the document, i.e. an absolute URL without the protocol or domain name
- the HTTP protocol version

1. Subsequent lines represent an HTTP header, giving the server information about what type of data is appropriate (e.g., what language, what MIME types), or other data altering its behavior (e.g., not sending an answer if it is already cached). These HTTP headers form a block which ends with an empty line.
2. The final block is an optional data block, which may contain further data mainly used by the POST method.

Structure of a server response
After the connected agent has sent its request, the web server processes it, and ultimately returns a response. Similar to a client request, a server response is formed of text directives, separated by CRLF, though divided into three blocks:

1. The first line, the status line, consists of an acknowledgment of the HTTP version used, followed by a status request (and its brief meaning in human-readable text).
2. Subsequent lines represent specific HTTP headers, giving the client information about the data sent (e.g. type, data size, compression algorithm used, hints about caching). Similarly to the block of HTTP headers for a client request, these HTTP headers form a block ending with an empty line.
3. The final block is a data block, which contains the optional data.

- **Able to demonstrate how to create an HTTP request**

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.

GET
The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

HEAD
The HEAD method asks for a response identical to that of a GET request, but without the response body.

POST
The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.

PUT
The PUT method replaces all current representations of the target resource with the request payload.

DELETE
The DELETE method deletes the specified resource.

CONNECT
The CONNECT method establishes a tunnel to the server identified by the target resource.

OPTIONS
The OPTIONS method is used to describe the communication options for the target resource.

TRACE
The TRACE method performs a message loop-back test along the path to the target resource.

PATCH
The PATCH method is used to apply partial modifications to a resource.

- **Able to recognize a REST endpoint**
- **Able to send or receive data either in body or in URL params**
- **Able recognize the status for the http request (success / failure)**

(See above HTTP status codes)