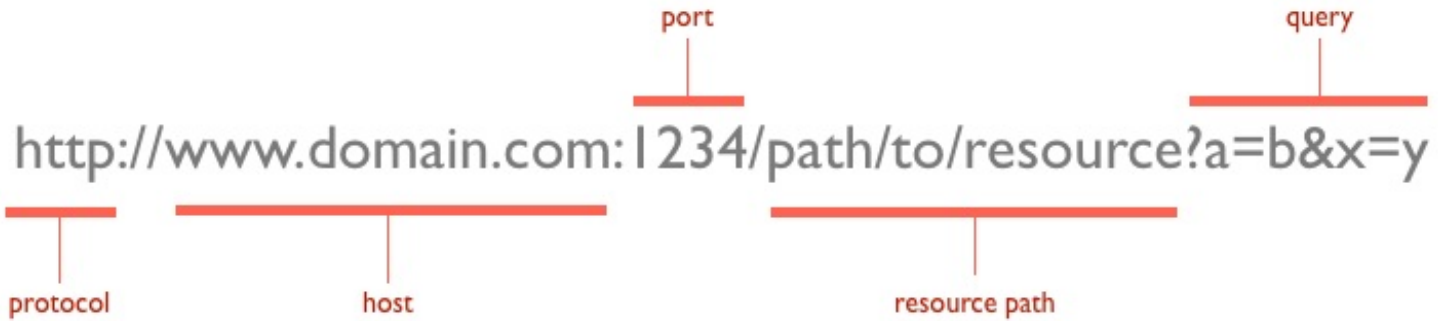
Lec 16 - Web APIs

Statistical Programming

Sta 323 | Spring 2022

Dr. Colin Rundel

URLs



From HTTP: The Protocol Every Web Developer Must Know

Query Strings

Provides named parameter(s) and value(s) that modify the behavior of the resulting page.

Format generally follows:

field1=value1&field2=value2&field3=value3

Some quick examples,

- `http://lmgtfy.com/?q=hello%20world`
- `http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address=1600+Amphitheatre+Parkway`
- `https://nomnom-prod-api.dennys.com/mapbox/geocoding/v5/mapbox.places/raleigh,%20nc.json?`

URL encoding

This is will often be handled automatically by your web browser or other tool, but it is useful to know a bit about what is happening

- Spaces will encoded as '+' or '%20'
- Certain characters are reserved and will be replaced with the percent-encoded version within a URL

!	#	$	&	'	(	)
%21	%23	%24	%26	%27	%28	%29
*	+	,	/	:	;	=
%2A	%2B	%2C	%2F	%3A	%3B	%3D
?	@	[	]			
%3F	%40	%5B	%5D			

- Characters that cannot be converted to the correct charset are replaced with HTML

```
URLencode("http://lmgtfy.com/?q=hello world")
```

```
## [1] "http://lmgtfy.com/?q=hello%20world"
```

```
URLdecode("http://lmgtfy.com/?q=hello%20world")
```

```
## [1] "http://lmgtfy.com/?q=hello world"
```

```
URLencode("! # $ & ' ( ) * + , / : ; = ? @ [ ]")
```

```
## [1] "!%20#%20$%20&%20'%20(%20)%20*%20+%20,%20/%20:%20;%20=%20?%20@%20[%20]"
```

```
URLdecode(URLencode("! # $ & ' ( ) *  + , / : ; = ? @ [ ]"))
```

```
## [1] "! # $ & ' ( ) *  + , / : ; = ? @ [ ]"
```

```
URLencode("Σ")
```

```
## [1] "%CE%A3"
```

```
URLdecode("%CE%A3")
```

```
## [1] "Σ"
```

RESTful APIs

REST

Representational **s**tate **t**ransfer

- describes an architectural style for web services (not a standard)
- all communication via http requests
- Key features:
  - addressible
  - stateless
  - connected
  - simple

HTTP Methods / Verbs

- GET - fetch a resource.
- POST - create a new resource.
- PUT - update a resource.
- DELETE - delete a resource.

Less common verbs: HEAD, TRACE, OPTIONS.

Based on HTTP: The Protocol Every Web Developer Must Know

Status Codes

- 1xx: Informational Messages
- 2xx: Successful
- 3xx: Redirection
- 4xx: Client Error
- 5xx: Server Error

Example 1:
An API of Ice And Fire

Documentation

While there is a lot of standardization, every API is different and you will need to review the documentation of each.

See documentation here for AAOIF.

Resources / endpoints:

- Root - `https://www.anapioficeandfire.com/api`
- List books - `https://www.anapioficeandfire.com/api/books`
- Specific book - `https://www.anapioficeandfire.com/api/books/1`

...

Pagination

> An API of Ice And Fire provides a lot of data about the world of Westeros. To prevent our servers from getting cranky, the API will automatically paginate the responses. You will learn how to create requests with pagination parameters and consume the response.
>
> **Things worth noting**:
> Information about the pagination is included in the Link header Page numbering is 1-based You can specify how many items you want to receive per page, the maximum is 50
>
> **Constructing a request with pagination**
> You specify which page you want to access with the ?page parameter, if you don't provide the ?page parameter the first page will be returned. You can also specify the size of the page with the ?pageSize parameter, if you don't provide the ?pageSize parameter the default size of 10 will be used.

Demo 1 - Basic access & pagination

Demo 2 - httr2 + headers

Exercise 1

Using the AAOIF API answer the following questions:

1. How many characters are included in this API?
2. What percentage of the characters are dead?
3. How many houses have an ancestral weapon?

Example 2: GitHub API