URLs

Stands for **Uniform Resource Locator**

A URL is a specific type of something known as a URI - **Uniform Resource Identifier**

One of the major properties of URLs is that they follow a **uniform system -** a standard format for presenting information

Secondly, they relate to resources - this definition is flexible, e.g. a website could be a resource and so could a database table etc

- even things not accessible over the internet could still be a resource we could identify

The system will then either identify or locate the resource.

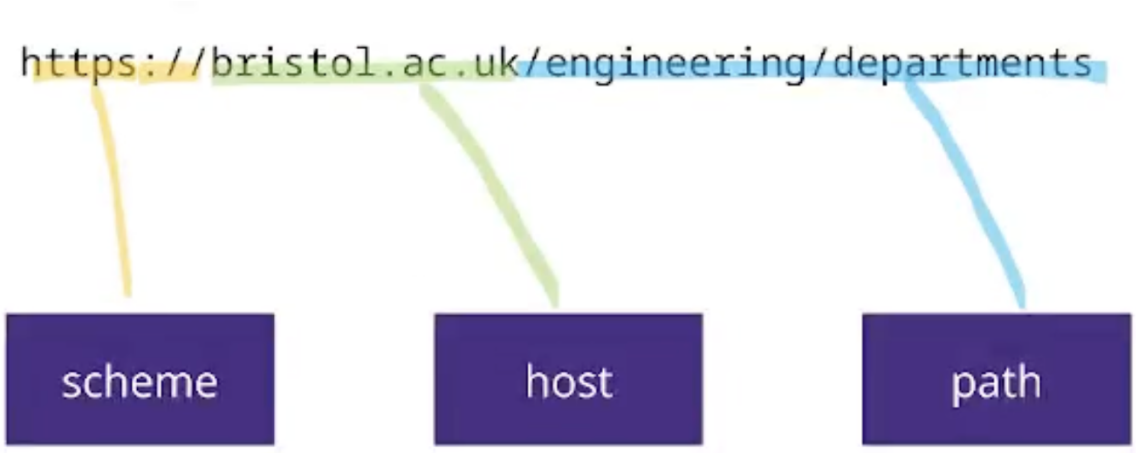
For an identification scheme, we dont requrie that using a URI will grant access to the resource, it could juts be used to refer to the resource.

A URL on the other hand is a type of URI where we want to locate the resource we presume that following the URL correctly will allow us to access that resource

```
https://bristol.ac.uk/engineering/
departments/computerscience
```

- this is a classic example of a URI which is also a URL consisting of 4 parts

URL Parts



Scheme - everything up to the colon is the scheme being used, the type of scheme used should help you interpret the rest of the URL

In the HTTP and HTTPS schemes the next part of the URL/I is the **host**

- this refers to the machine or service you want to connect to
- Optionally here you could also specify a port to connect to which would come at the end of the host, if none is specified you would use the default port for the protocol would be used

Finally you have the **path,** this is referring to a specific resource on a machine

- often but not always analogous to the directory structure on a web server
  - in http the path you specify in the URL will become the **argument to the** `GET` request that you make to the server
    - e.g. in the above example we would be connecting to bristol.ac.uk on the default https port ad we are sending a request to that server saying we want to `GET` /engineering/departments

URL Queries

- as a final part of URL syntax we can create queries

  - the first line here is what was spoken about above: scheme, name, path to resource

- we are the additionally adding a query that starts with `?` and is made up of parameters which will be passed to the resource
- in the above example we are passing two parameters: `name` which we are setting to the vaslue of welcome (`name=welcome`) and (`&`) the second is `action` which we are setting to the value of view (`action=view`)
    - these are passed as key-value pairs to the resource

Paremeters in HTTP style queries are serperated by ampersands (&)

so in the above example, we are accessing the resource called `pages` and as we are accessing it we are sending it a query

- What it does with the query parameters is based on how it has been designed, i.e. the internal logic of the `pages` resource and what it knows to do woth paramters that are passed to it
    - it could juts print these values on a webpage, or execute a database query for us

URI Schemes

```
ftp://ftp.is.co.za/rfc/rfc1808.txt
http://www.ietf.org/rfc/rfc2396.txt
ldap://[2001:db8::7]/c=GB?objectClass?one
mailto:John.Doe@example.com
news:comp.infosystems.www.servers.unix
tel:+1-816-555-1212
telnet://192.0.2.16:80/
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

- these refer to different kinds of resources that you would'nt typically access through a browser
-

REST

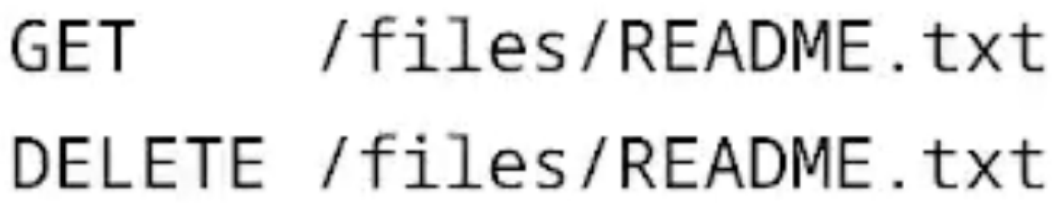
Representational State Transfer

- refers to an **architectural philosophy** that the web was mean to be built around

as discussed, HTTP is stateless. If you need to carry any state it should be **held in the request to the server**

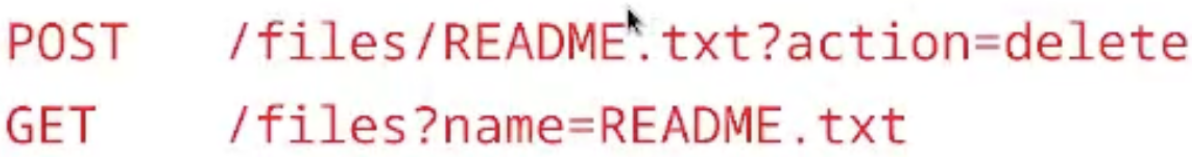
- state should be held with the client not the server
- Resources should have names: **URLs**
- We should interact with these resources via their names using using HTTP **methods**
  - i.e. verbs such as `GET` `PUT` etc etc

In theory much of the web's architecure was built to support these principles but in practise some websites do not always follow it...



- REST principles would have it so that if you wanted to get a file (README.txt) from a server, you would issue a `GET` request for that resource
- say logic for if you wanted to delete that file - issue a `DELETE` request for that resource

In practise however developers have created systems such as these, where **queries** are used to define these sorts of behaviour:



- People `POST` a request for a resource with a *query* parameter for *deletion*
- or getting a resource called files with a name parameter for the file that you want, e.g.

```
GET /files?name=README.txt
```

The issue with systems like this are they are not standadised

- queries in other resources else where may look different for getting or deleting a file
- therefore you end up having to learn a new system for interacting with a new domain - this is inefficient!

URL Fragments

- optional part at the end of a URL which starts with `#` and it lets you refer to specific parts of the document you have accessed

Why use them?

- when sourcing info you can link directly to the quote or content you have cited
-