Edinburgh Napier UNIVERSITY

ADVANCED WEB TECHNOLOGIES
SET09103
TOPIC #05

STATELESSNESS & STATE

Dr Simon Wells
s.wells@napier.ac.uk
http://www.simonwells.org

TL/DR

- HTTP is meant to be stateless protocol, but a lot of effort has been put into making things more stateful. Let's explore that…

Edinburgh Napier UNIVERSITY

OVERVIEW

- Part# 1: Statelessness & statefulness
- Part#2: Examples…

Edinburgh Napier
UNIVERSITY

PART #1: BACKGROUND

Edinburgh Napier UNIVERSITY

STATELESS

- We've established earlier that one of the properties of HTTP is that it is a **stateless** protocol.
  - The server doesn't *maintain state* across multiple request-response cycles.
- Essentially means that the server need not do a lot of work to respond to the request.
- Simplest case: Every page is an individual HTML file stored on disk. A request retrieves the file and sends it back to the client. Client then retrieves any related resources.
- Advantage: Very efficient & Very Scalable
- Disadvantage: ???

Edinburgh Napier UNIVERSITY

STATEFULNESS

- Developers have tried to get around the statelessness property for almost as long as it has existed. There are several ways to do so, e.g. query parameters, header entries, cookies.

- If the server must keep track of the *state* of the world between request-response cycles then we are making the design more *stateful*.

- This might be to fulfil some interaction, UX, or design need.

- Why might we need to maintain state between HTTP cycles?

- We are focussing more on dynamic server-side functionality, but does statefulness always imply dynamic server-side functionality?

Edinburgh Napier UNIVERSITY

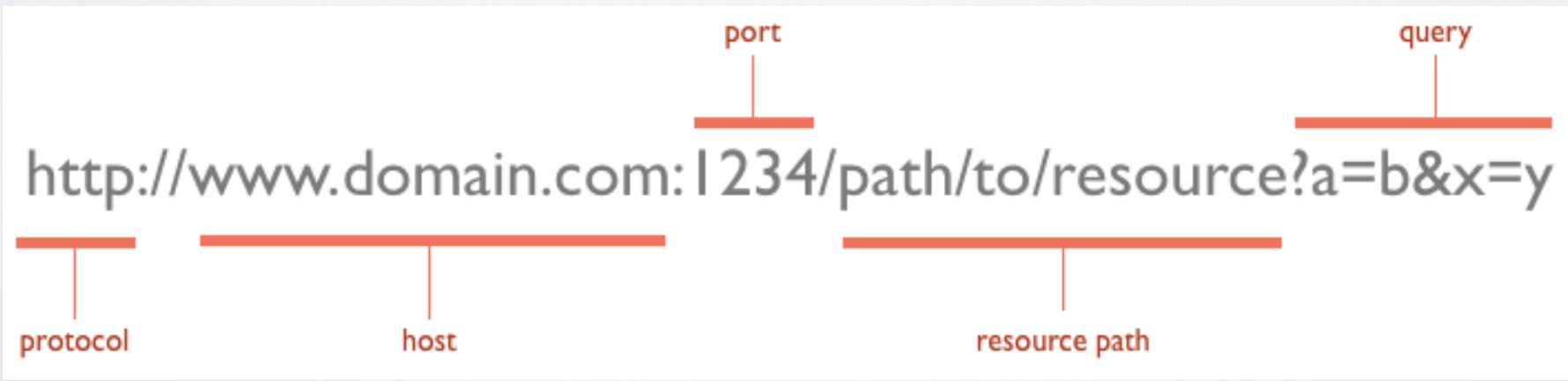
TRANSIENT DATA

- Today we'll mostly consider *transient data* as opposed to *permanent data*.
- Not hard and fast, or even clear distinction but useful to think of:
  - **Transient data:** needed for a shorter period of time, i.e. access tokens, log-in credentials, shopping basket, personalised settings, …
  - **Permanent data:** stored in a database for the longer term. Might be further processed and aggregated. We'll consider this in the next topic.

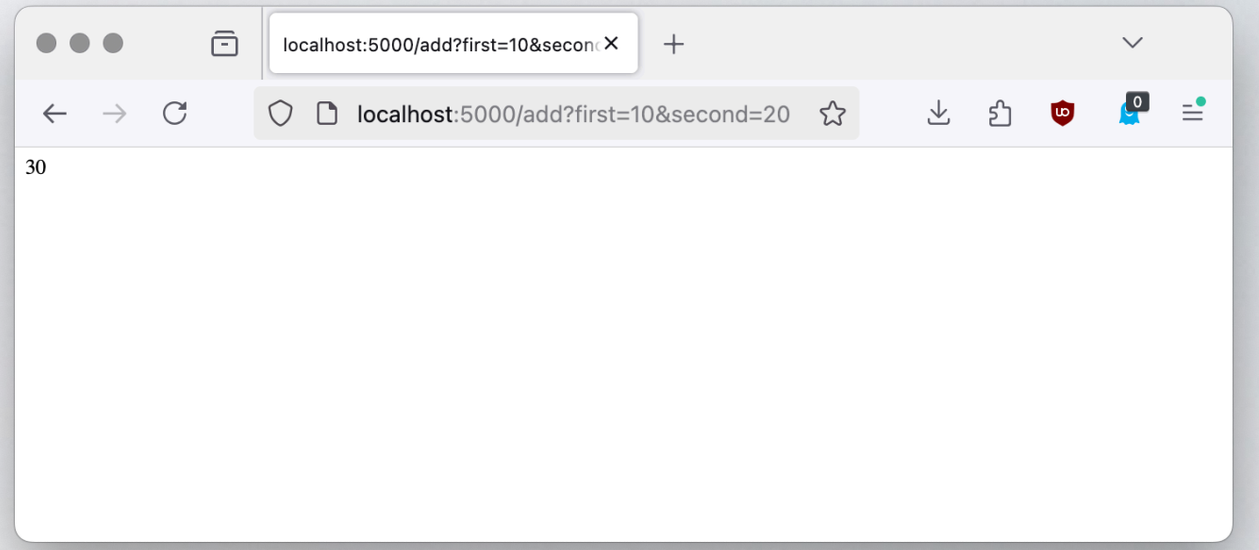
Edinburgh Napier UNIVERSITY

QUERY PARAMETERS

- Fundamental part of the structure of URLs.
- Add information into the address after identifying the resource path.
- Supports arbitrary ampersand separated key:value pairs.
- Both keys and values must be URL-encoded.
- Low barrier to entry but quickly becomes unwieldy for anything more than a handful of paremeters.
- Can make the URL and associated structures very difficult to manipulate.



QUERY EXAMPLE




```
from flask import Flask, request
app = Flask(__name__)


@app.route("/add")
def add():
    first = int(request.args.get('first'))
    second = int(request.args.get('second'))
    return str(first+second)
```

HEADERS

- Recall that HTTP requests and responses both have a similar structure made up of a head and a body.
- Headers can be used to communicate additional information between the client & server. Useful if it doesn't need to be immediately accessible.
- Can use custom (or standard) headers to communicate backwards and forwards.
- Like URL Query Parameters, headers are strings containing key:value pairs and have similar limitations but are useful for information that is conceptually more part of the protocol.

Request Header:

```
GET / HTTP/1.1
Host: www.msaleh.co.cc
User-Agent:Mozilla/5.0 (Windows; U; Windows
NT 5.1; en-US; rv:1.9.0.10)
Gecko/2009042316 Firefox/3.0.10
Accept: */*
Connection:close
```

Response Header:

```
HTTP/1.1 200 OK
Date: Sat, 09 May 2009 12:27:54 GMT
Server: Apache/2.2.11 (Unix)
Last-Modified: Thu, 12 Feb 2009 15:29:42
GMT
Etag: "c3b-462ba63a46580"-gzip
Cache-Control: max-age=1200, private,
proxy-revalidate, must-revalidate
Expires: Sat, 09 May 2009 12:47:54 GMT
Accept-Ranges: bytes
Content-Length: 976
Content-Type:text/html
```

Edinburgh Napier
UNIVERSITY

HEADER EXAMPLE

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def index():
    user_agent = request.headers.get('User-Agent')
    return f"User-Agent: {user_agent}"
```

Edinburgh Napier UNIVERSITY

COOKIES

- Cookies are kind of infamous due to their misuse.
- But are very relevant, useful, and fundamental to modern web development (if we want dynamic stateful sites).
- Cookies are again strings, and are actually passed to the server in a header line as part of the HTTP communications.
- Stored by the browser between HTTP cycles.
- Implemented in Flask as part of the **sessions** functionality.

Edinburgh Napier
UNIVERSITY

FLASK SESSIONS

- The user interacts with the flask server multiple times.
- These interactions form a **session** made up of multiple HTTP request-response cycles.
- Sessions aren't a formal part of HTTP but are a useful concept for grouping together the functionality for implementing stateful HTTP interactions.
- Session data is stored in a cookie, which is automatically passed backwards and forwards between the client and server
- Cookie initially created by the Flask app (server-side), and passed to the client for storage between session.
- Client passes the cookie to the server with each request.
- Server processes the cookie, extracts and uses what it needs, processes the request, and returns the response with an associated, possibly updated, cookie.

FLASK PATTERN: MESSAGE FLASHING

Edinburgh Napier UNIVERSITY

- Related to the idea of sharing information between HTTP cycles.
- Sometimes we want to be able to submit some information and rather than a dedicated result page, just get some information back about the success or failure of the last action.

FLASK PATTERN: MESSAGE FLASHING

```
from flask import Flask, flash, redirect, request, url_for, render_template

app = Flask(__name__)
app.secret_key = 'supersecret'

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login/')
@app.route('/login/<message>')
def login(message=None):
    if (message != None):
        flash(message)
    else:
        flash(u'A default message')
    return redirect(url_for('index'))
```

```
<html>
<body>
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <ul>
    {% for message in messages %}
      <li>{{ message | safe }}</li>
    {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
</body>
</html>
```

Edinburgh Napier UNIVERSITY

LOGGING

- Things are getting more complex now.
- Logging is practically important:
  - Need to know what is happening when our app runs
  - Might want to keep track of information in a structured way
  - Rather than *ad hoc* print debugging or running a debugger to try to recreate an issue.

```
import configparser
import logging

from logging.handlers import RotatingFileHandler
from flask import Flask, url_for

app = Flask(__name__)

@app.route('/')
def root():
        this_route = url_for('.root')
        app.logger.info("Logging a test message from "+this_route)
        return "Hello Napier from the configuration testing app (Now with added logging)"

def init(app):
    config = configparser.ConfigParser()
    try:
        config_location = "etc/logging.cfg"
        config.read(config_location)
        app.config['log_file'] = config.get("logging", "name")
        app.config['log_location'] = config.get("logging", "location")
        app.config['log_level'] = config.get("logging", "level")
    except:
        print("Could not read configs from: ", config_location)

def logs(app):
    log_pathname = app.config['log_location'] + app.config['log_file']
    file_handler = RotatingFileHandler(log_pathname, maxBytes=1024* 1024 * 10 , backupCount=1024)
    file_handler.setLevel( app.config['log_level'] )
    formatter = logging.Formatter("%(levelname)s | %(asctime)s |  %(module)s | %(funcName)s | %(message)s")
    file_handler.setFormatter(formatter)
    app.logger.setLevel( app.config['log_level'] )
    app.logger.addHandler(file_handler)

init(app)
logs(app)
```

Edinburgh Napier UNIVERSITY

TESTING

- This goes hand-in-hand with logging.

- As your code becomes larger and more complex

- You will start to get nervous about where any given change will unexpectedly break previously good code.

TESTING EXAMPLE

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def root():
  return "HELLO NAPIER", 200
```

```
import unittest
import testing

class TestingTest(unittest.TestCase):
  def test_root(self):
    self.app = testing.app.test_client()
    out = self.app.get('/')
    assert '200 OK' in out.status
    assert 'charset=utf-8' in out.content_type
    assert 'text/html' in out.content_type
```

Edinburgh Napier UNIVERSITY

SUMMARY

- We've considered why we might want to maintain state rather than run with the default statelessness of HTTP.

- We've surveyed a range of methods that we can use to maintain state.

Edinburgh Napier
UNIVERSITY

PART #2: PRACTICAL WORK

Edinburgh Napier UNIVERSITY

WRAPPING UP

- We've taken a whistle-stop tour of state and statelessness in modern web development:

- There's lot's more to see & We'll be returning to some of these topics in more detail later, e.g. data and databases.