AARHUS UNIVERSITET

Software Engineering and Architecture

Hyper Text Transfer Protocol

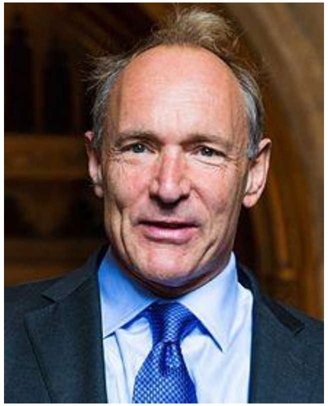
HTTP

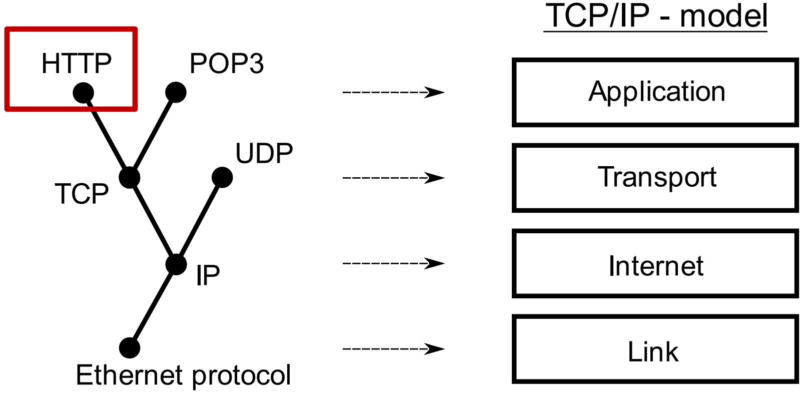
WWW

- Tim Berners-Lee approx. 1989 - 1990
  - Task: Sharing research documents at CERN

- Solution:
  - Hypertext protocol over TCP/IP for retrieving documents

- Actually very simple text based format





Just a Note

- Web, world wide web, HTML, HTTP may seem like one big jumble but they are *distinct concepts* though they were developed in parallel. They have different *roles* to play.
  - – HTML: Hypertext Markup Language is a **dataformat**, useful for visual formatting of text document containing images and references (hyperlinks) to other documents.
  - – HTTP: Hypertext Transfer Protocol is an **application protocol** for distributed information systems. (Actually not related to 'hypertext' ☺)
  - – WWW: The internet-based TCP/IP **system** made that used HTML+HTTP to share documents at CERN, and later – quite a few other places ☺

Request-Reply Protocol

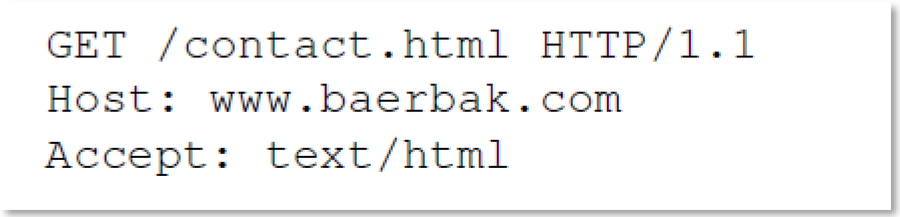
- As ‘I want to view this HTML document’ is essentially a *synchronous* task…
  - No point in reading a document that is not loaded yet…

- HTTP adhere to the *request-reply protocol*
  - My browser sends a request to a web server **and blocks until…**
  - … the server has returned a document, after which…
  - My browser renders the text on my screen…
- HTTP thus must define
  - Format of request. Format of reply.

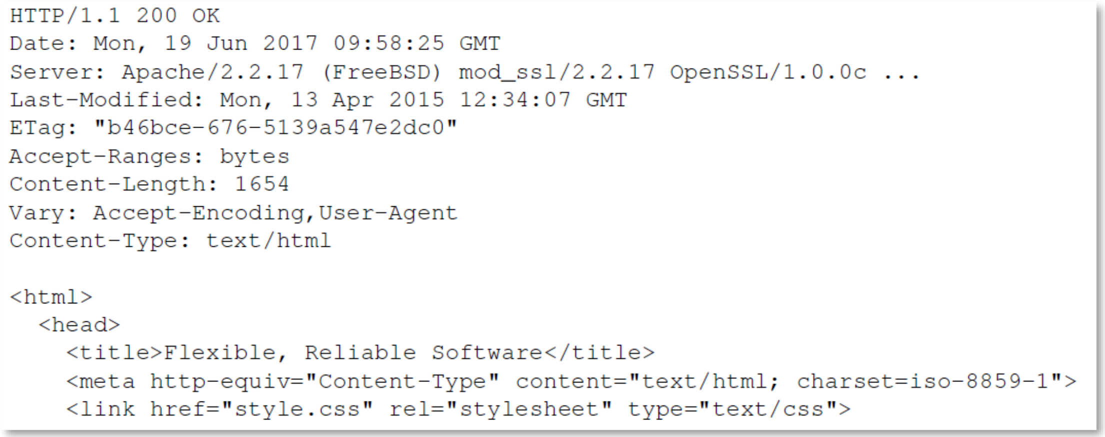
“Marshalling” - Message Format

Text format !

- Request line
  - Verb resource HTTP version
  - Header key-values
  - *Empty line*
  - (contents)
- Reply line
  - Status line
    - HTTP codes
  - Header fields
  - *Empty line*
  - Message body



```
GET /contact.html HTTP/1.1
Host: www.baerbak.com
Accept: text/html
```

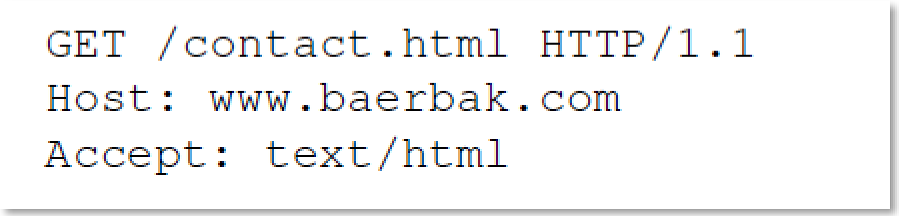


```
HTTP/1.1 200 OK
Date: Mon, 19 Jun 2017 09:58:25 GMT
Server: Apache/2.2.17 (FreeBSD) mod_ssl/2.2.17 OpenSSL/1.0.0c ...
Last-Modified: Mon, 13 Apr 2015 12:34:07 GMT
ETag: "b46bce-676-5139a547e2dc0"
Accept-Ranges: bytes
Content-Length: 1654
Vary: Accept-Encoding,User-Agent
Content-Type: text/html

<html>
  <head>
    <title>Flexible, Reliable Software</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link href="style.css" rel="stylesheet" type="text/css">
```

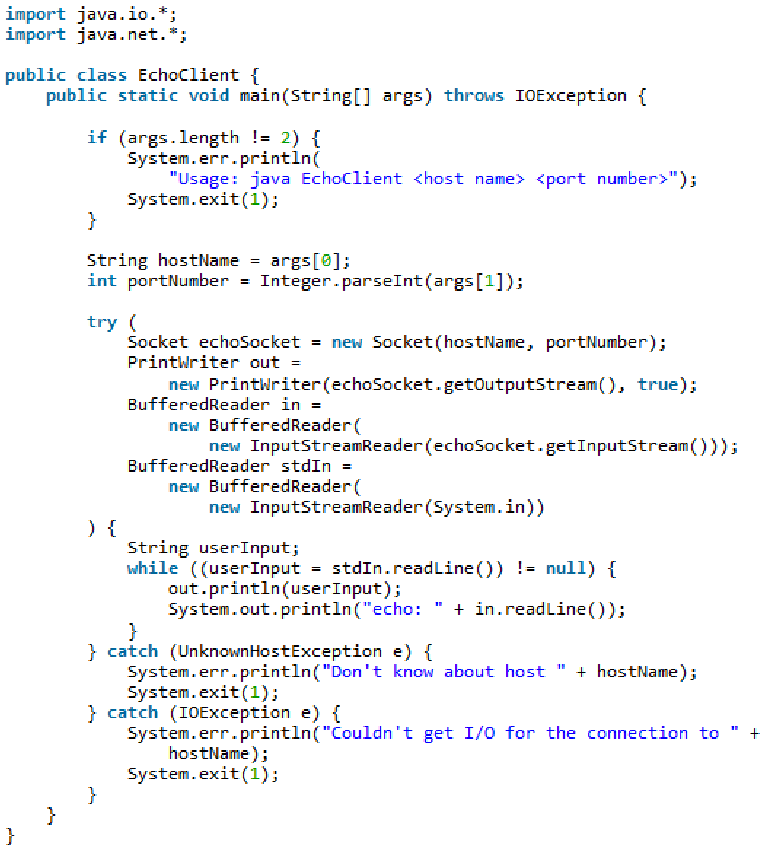
Write your Own Web Client

- Exercise in class:
  - Write a web client

“java webclient www.baerbak.com”



```
GET /contact.html HTTP/1.1
Host: www.baerbak.com
Accept: text/html
```

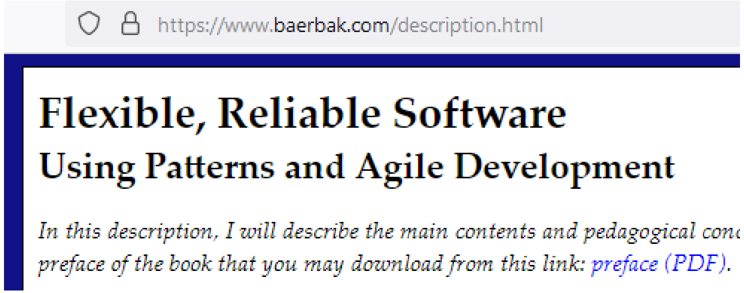


```
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {

        if (args.length != 2) {
            System.err.println(
                "Usage: java EchoClient <host name> <port number>");
            System.exit(1);
        }

        String hostName = args[0];
        int portNumber = Integer.parseInt(args[1]);

        try (
            Socket echoSocket = new Socket(hostName, portNumber);
            PrintWriter out =
                new PrintWriter(echoSocket.getOutputStream(), true);
            BufferedReader in =
                new BufferedReader(
                    new InputStreamReader(echoSocket.getInputStream()));
            BufferedReader stdIn =
                new BufferedReader(
                    new InputStreamReader(System.in))
        ) {
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
            }
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host " + hostName);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection to " +
                hostName);
            System.exit(1);
        }
    }
}
```

AARHUS UNIVERSITET

Clients

- The most well-known clients are *browsers*



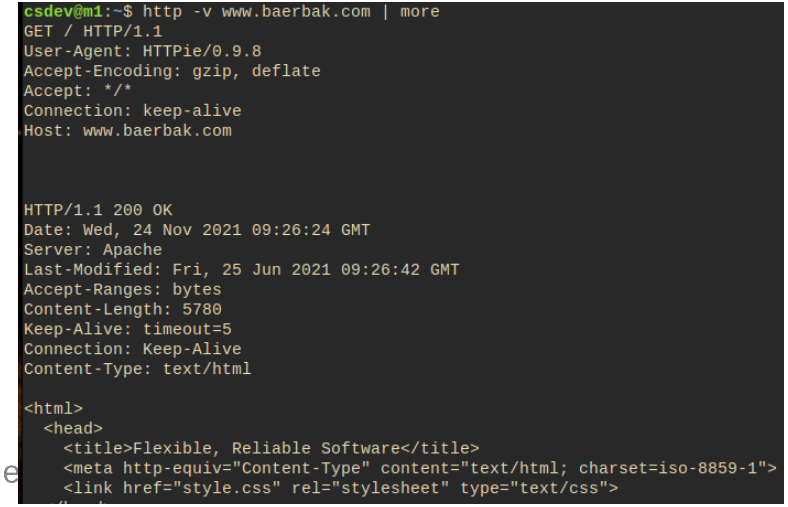
https://www.baerbak.com/description.html

Flexible, Reliable Software

Using Patterns and Agile Development

In this description, I will describe the main contents and pedagogical conc
preface of the book that you may download from this link: preface (PDF).

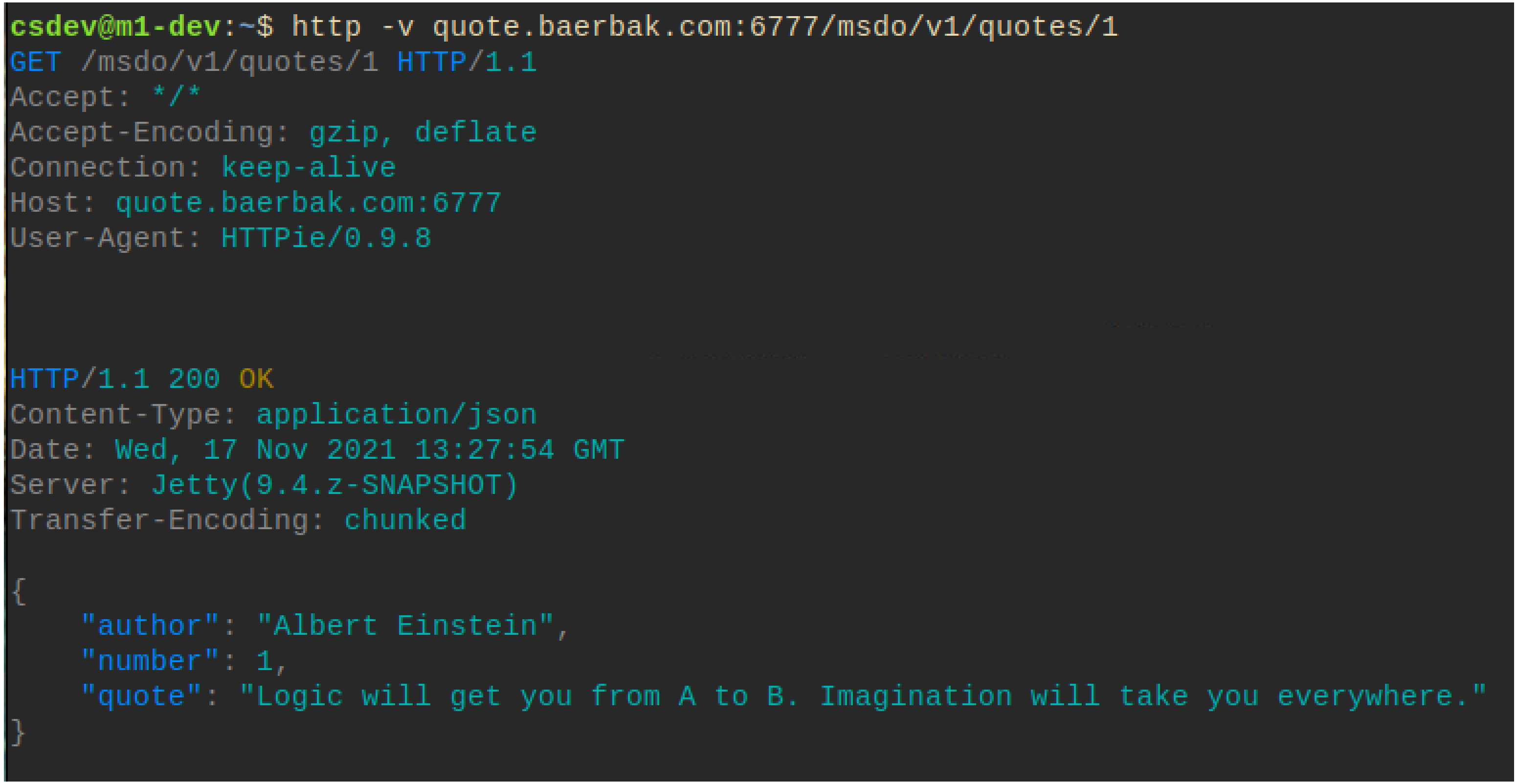
- Developers often use *commandline* browsers instead
  - curl (= see-url)
  - httpie



```
csdev@m1:~$ http -v www.baerbak.com | more
GET / HTTP/1.1
User-Agent: HTTPie/0.9.8
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Host: www.baerbak.com



HTTP/1.1 200 OK
Date: Wed, 24 Nov 2021 09:26:24 GMT
Server: Apache
Last-Modified: Fri, 25 Jun 2021 09:26:42 GMT
Accept-Ranges: bytes
Content-Length: 5780
Keep-Alive: timeout=5
Connection: Keep-Alive
Content-Type: text/html

<html>
  <head>
    <title>Flexible, Reliable Software</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <link href="style.css" rel="stylesheet" type="text/css">
```

Ala…

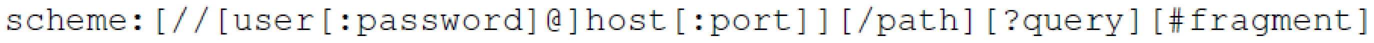
- Testing the quote service for my ‘MicroService’ course



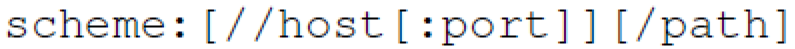
```
csdev@m1-dev:~$ http -v quote.baerbak.com:6777/msdo/v1/quotes/1
GET /msdo/v1/quotes/1 HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: quote.baerbak.com:6777
User-Agent: HTTPie/0.9.8



HTTP/1.1 200 OK
Content-Type: application/json
Date: Wed, 17 Nov 2021 13:27:54 GMT
Server: Jetty(9.4.z-SNAPSHOT)
Transfer-Encoding: chunked

{
    "author": "Albert Einstein",
    "number": 1,
    "quote": "Logic will get you from A to B. Imagination will take you everywhere."
}
```

URI / URL

- HTTP is about *resources = named data/information*
  - Naming the resources follows a strict format
- URI: Uniform Resource Identifier



```
scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
```



```
scheme:[//host[:port]][/path]
```

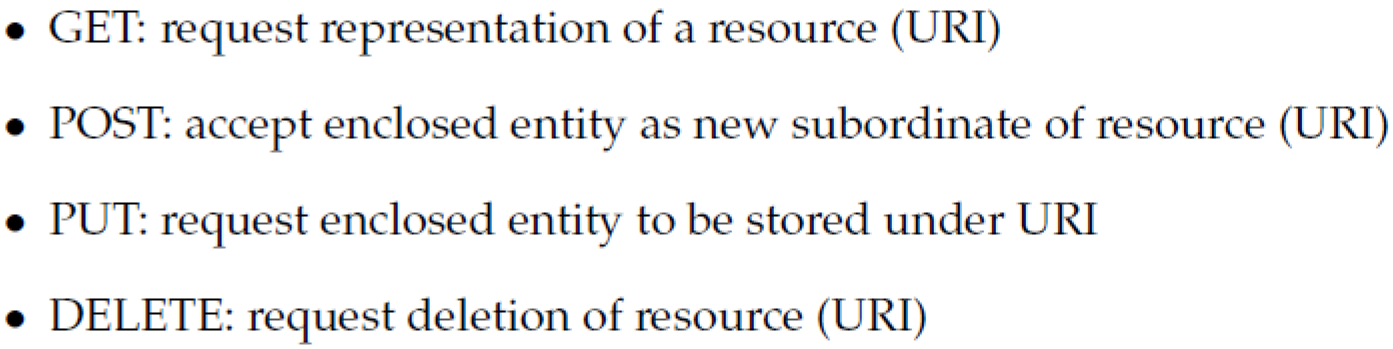
- URL = URI in which resource location and means are defined
  - [**http**://www.baerbak.com/contact.html](http://www.baerbak.com/contact.html)
  - **http://localhost:4567/bin**

Exercise:
Identify the parts of the URI

AARHUS UNIVERSITET

HTTP Verbs

- Http version 1.1. defines 4 verbs (ok, some more…)



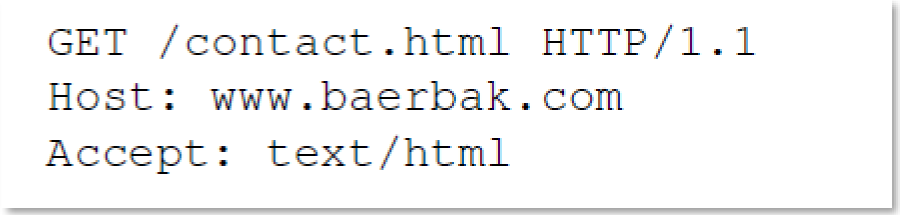
- GET: request representation of a resource (URI)
- POST: accept enclosed entity as new subordinate of resource (URI)
- PUT: request enclosed entity to be stored under URI
- DELETE: request deletion of resource (URI)

- … which are basically the **database verbs**
  - **CRUD Create, Read, Update, Delete**

- ***These form the core of the REST architectural style…***

GET

- GET is the ‘first and original verb’, and the one most traffic uses on WWW
  - Browing web pages



```
GET /contact.html HTTP/1.1
Host: www.baerbak.com
Accept: text/html
```

  - Or even make searches on the web server

```
scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
```

- GET is idempotent
  - Call once or 100 times, the output is the same
  - **It is an ‘accessor’ / ‘query’ method!**

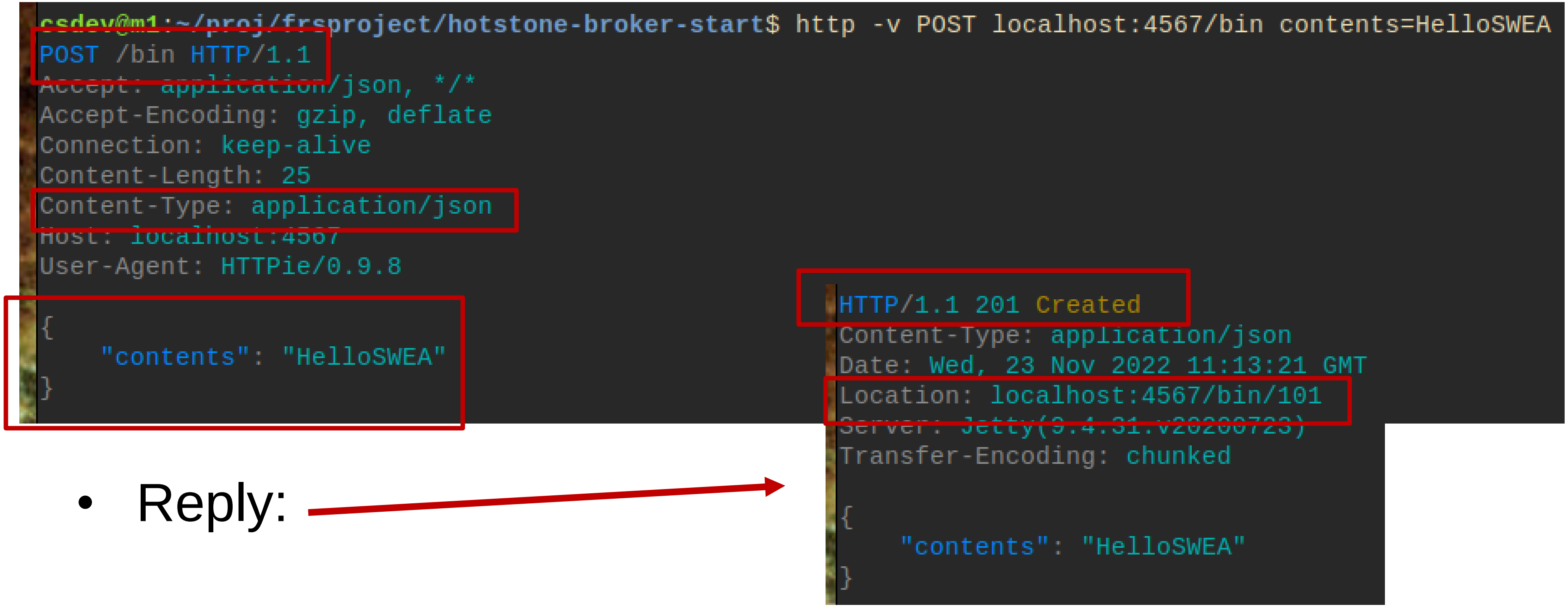
POST

- POST means ‘create’
  - *That is, create new resources/information on the server*
  - **It is a ‘mutator’/’command’ method

- Consider ‘paystation.addPayment(5);’
  - Command pattern: *Convert method call to an object*
- *Now, consider that ‘paystation’ is on the server side*
  - POST allows us to ***create a command object***
    - POST /paystation HTTP/1.1
    - Body { method: ‘addPayment’, argument: ‘5’ }

POST Example

- Example: A ‘Pastebin’ server accepting contents on its /bin path. Contents encoded as JSON.



```
csdev@m1:~/proj/frsproject/hotstone-broker-start$ http -v POST localhost:4567/bin contents=HelloSWEA
POST /bin HTTP/1.1
Accept: application/json, */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 25
Content-Type: application/json
Host: localhost:4567
User-Agent: HTTPie/0.9.8

{
    "contents": "HelloSWEA"
}
```

- Reply:

```
HTTP/1.1 201 Created
Content-Type: application/json
Date: Wed, 23 Nov 2022 11:13:21 GMT
Location: localhost:4567/bin/101
Server: Jetty(9.4.31.v20200723)
Transfer-Encoding: chunked

{
    "contents": "HelloSWEA"
}
```

PUT, DELETE

- ... Will we return to later, when we discuss REST...
- PUT = update existing information
- DELETE = (guess ☺)
- CRUD = Create, Read, Update, Delete
  - HTTP is basically a database protocol on shared resources ☺

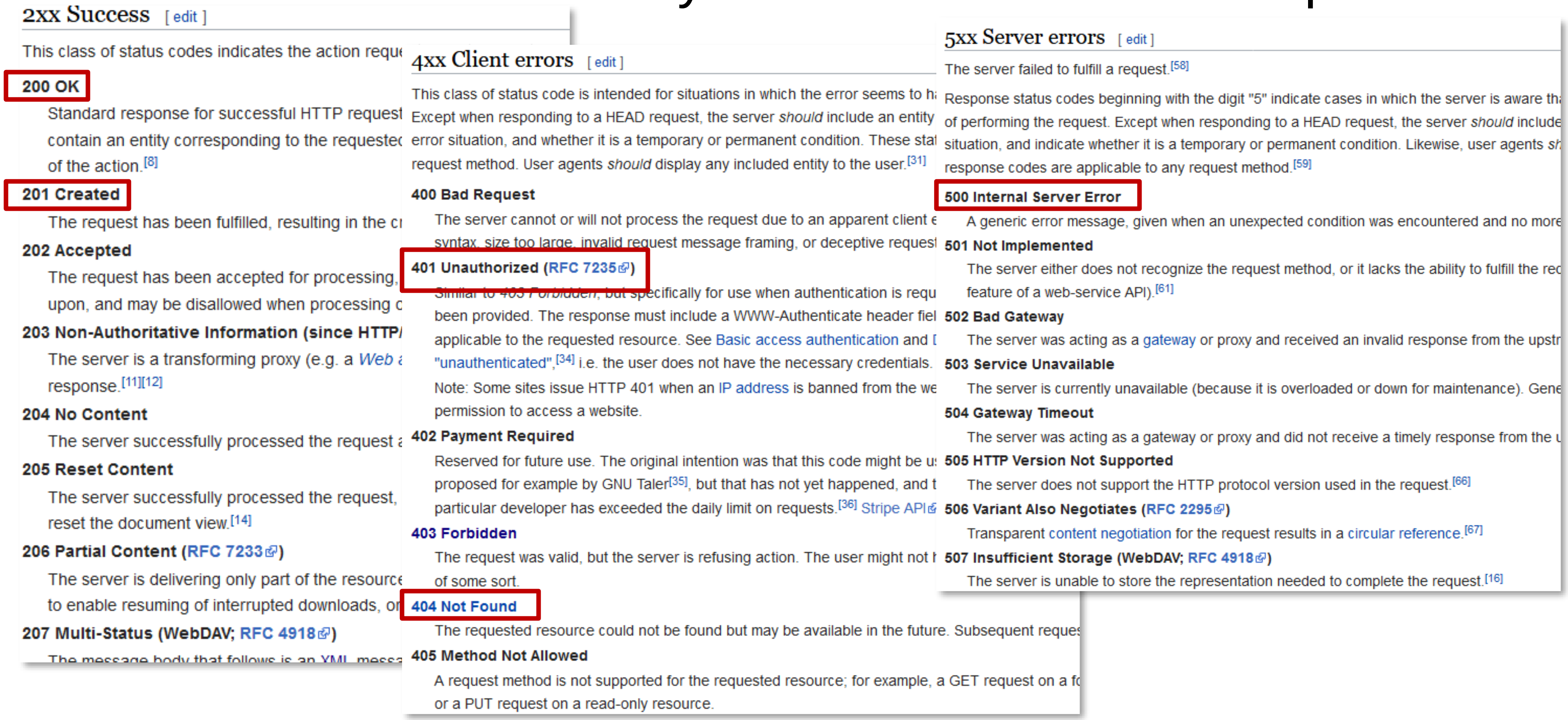
Failures in Distribution

- A lot of things can and will go wrong in distributed systems
  - The server has crashed
  - The network has crashed
  - Server does not understand what you talk about
  - You do not have the proper authorization
- We normally use *exceptions* to signal failures
- But – does not work over networks ☹

- The old way: **Error codes**

HTTP Status Codes

- Well defined vocabulary of error codes! See Wikipedia



2xx Success [edit]

This class of status codes indicates the action reque

200 OK

Standard response for successful HTTP request contain an entity corresponding to the requested of the action.[8]

201 Created

The request has been fulfilled, resulting in the cr

202 Accepted

The request has been accepted for processing, upon, and may be disallowed when processing o

203 Non-Authoritative Information (since HTTP/

The server is a transforming proxy (e.g. a *Web a* response.[11][12]

204 No Content

The server successfully processed the request a

205 Reset Content

The server successfully processed the request, reset the document view.[14]

206 Partial Content (RFC 7233)

The server is delivering only part of the resource to enable resuming of interrupted downloads, or

207 Multi-Status (WebDAV; RFC 4918)

The message body that follows is an XML messa

4xx Client errors [edit]

This class of status code is intended for situations in which the error seems to ha Except when responding to a HEAD request, the server *should* include an entity error situation, and whether it is a temporary or permanent condition. These sta request method. User agents *should* display any included entity to the user.[31]

400 Bad Request

The server cannot or will not process the request due to an apparent client e syntax, size too large, invalid request message framing, or deceptive request

401 Unauthorized (RFC 7235)

Similar to *403 Forbidden*, but specifically for use when authentication is requ been provided. The response must include a WWW-Authenticate header fiel applicable to the requested resource. See Basic access authentication and D "unauthenticated",[34] i.e. the user does not have the necessary credentials.

Note: Some sites issue HTTP 401 when an IP address is banned from the we permission to access a website.

402 Payment Required

Reserved for future use. The original intention was that this code might be us proposed for example by GNU Taler[35], but that has not yet happened, and t particular developer has exceeded the daily limit on requests.[36] Stripe API

403 Forbidden

The request was valid, but the server is refusing action. The user might not h of some sort.

404 Not Found

The requested resource could not be found but may be available in the future. Subsequent reques

405 Method Not Allowed

A request method is not supported for the requested resource; for example, a GET request on a fo or a PUT request on a read-only resource.

5xx Server errors [edit]

The server failed to fulfill a request.[58]

Response status codes beginning with the digit "5" indicate cases in which the server is aware tha of performing the request. Except when responding to a HEAD request, the server *should* include situation, and indicate whether it is a temporary or permanent condition. Likewise, user agents *sh* response codes are applicable to any request method.[59]

500 Internal Server Error

A generic error message, given when an unexpected condition was encountered and no more

501 Not Implemented

The server either does not recognize the request method, or it lacks the ability to fulfill the req feature of a web-service API).[61]

502 Bad Gateway

The server was acting as a gateway or proxy and received an invalid response from the upstr

503 Service Unavailable

The server is currently unavailable (because it is overloaded or down for maintenance). Gene

504 Gateway Timeout

The server was acting as a gateway or proxy and did not receive a timely response from the u

505 HTTP Version Not Supported

The server does not support the HTTP protocol version used in the request.[66]

506 Variant Also Negotiates (RFC 2295)

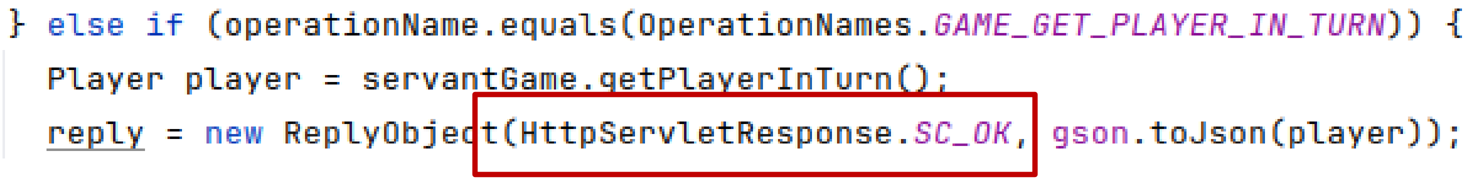
Transparent content negotiation for the request results in a circular reference.[67]

507 Insufficient Storage (WebDAV; RFC 4918)

The server is unable to store the representation needed to complete the request.[16]

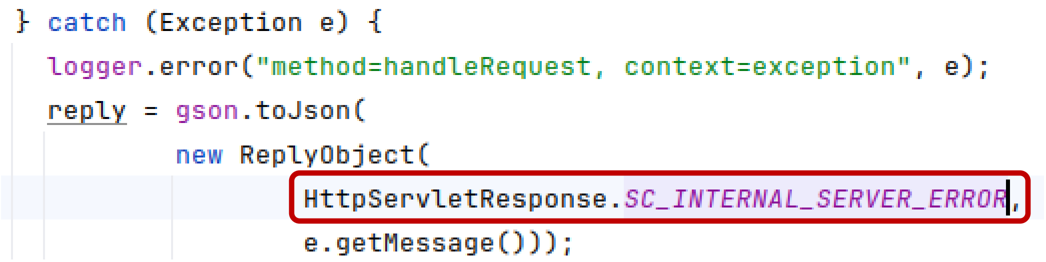
I have reused these in Broker

- You have probably already used these in Invoker code



```
} else if (operationName.equals(OperationNames.GAME_GET_PLAYER_IN_TURN)) {
  Player player = servantGame.getPlayerInTurn();
  reply = new ReplyObject(HttpServletResponse.SC_OK, gson.toJson(player));
```

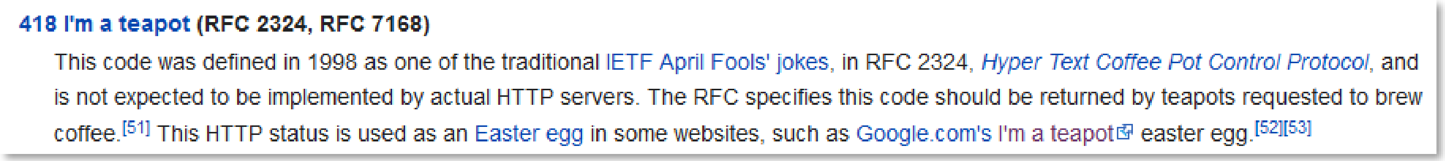
- And if you have added exception handling from TeleMed



```
} catch (Exception e) {
  logger.error("method=handleRequest, context=exception", e);
  reply = gson.toJson(
          new ReplyObject(
                  HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                  e.getMessage()));
```

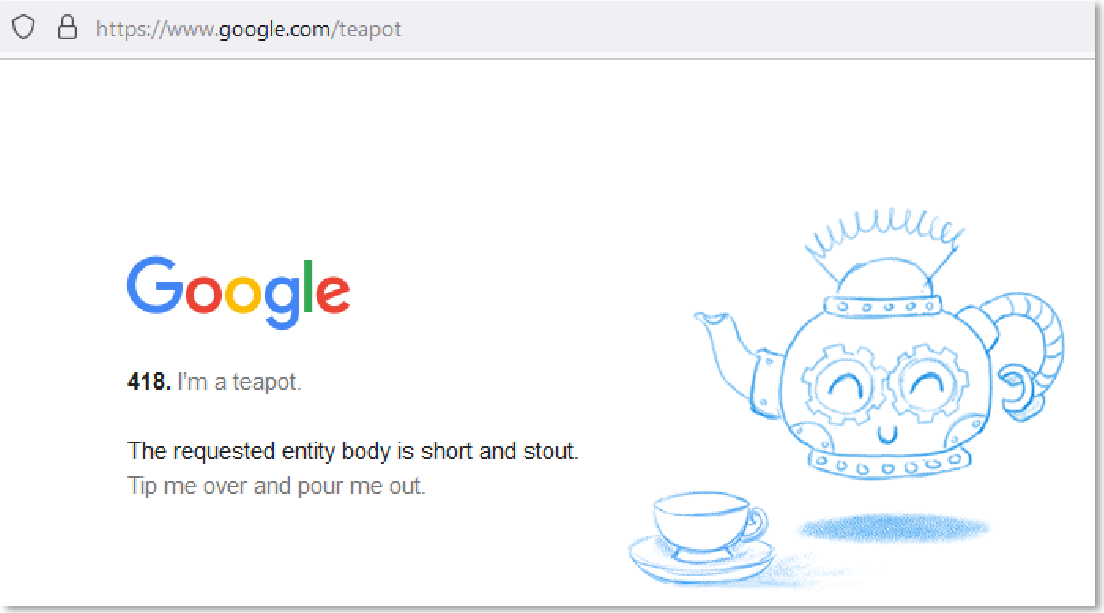
Fun Fact

- Code 418



418 I'm a teapot (RFC 2324, RFC 7168)

This code was defined in 1998 as one of the traditional IETF April Fools' jokes, in RFC 2324, *Hyper Text Coffee Pot Control Protocol*, and is not expected to be implemented by actual HTTP servers. The RFC specifies this code should be returned by teapots requested to brew coffee.[51] This HTTP status is used as an Easter egg in some websites, such as Google.com's I'm a teapot easter egg.[52][53]



https://www.google.com/teapot

418. I'm a teapot.

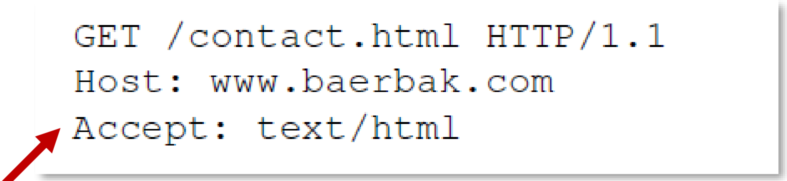
The requested entity body is short and stout.

Tip me over and pour me out.

Media Types

- The requestor and the replier need to agree on the *dataformat* that data is exchanged in
  - Media types, defined by IANA
    - Internet Assigned Number Authority
- Well known types

– text/html:	HTML formatted text
– image/gif:	Image in the GIF format
– application/xml:	XML format
– application/json:	JSON format



```
GET /contact.html HTTP/1.1
Host: www.baerbak.com
Accept: text/html
```

I want HTML, please

Summary

- HTTP is a protocol = interaction requirements
  - – Defining the contract between client and server roles
  - – Basically just exchange of text messages
    - Defined by a format ala
      - – Verb line (request) Status line (replies)
      - – Headers – (key,value) pairs
      - – Empty line
      - – “body” = the core contents of the message
  - – Verbs are GET, POST, PUT, DELETE
  - – Media types define data format of the ‘body’
  - – Status codes defines a vocabulary of error types
    - 200 OK and 404 Not Found