AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

REST Architectural Style

Henrik Bærbak Christensen

AARHUS UNIVERSITET

REST: The Fast Version

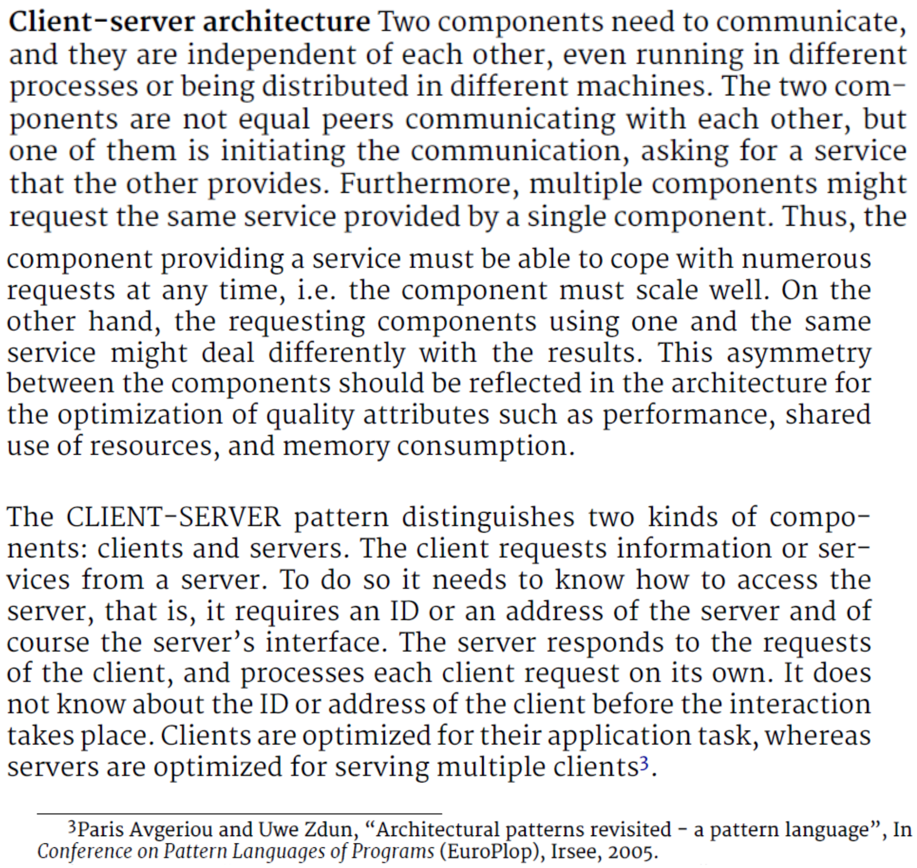
Assuming you already know REST ☺

Architectural Style

- As a software architect, I see REST as an
  - **Architectural style / pattern**
- It is a specific programming model
  - Functional programming:
    - Computation is passing data through chains of functions
  - Object programming:
    - Computation is community of objects passing messages
  - RPC over Client-Server:
    - Computation is clients invoking procedures on remote servers
  - REST
    - Computation is clients manipulating resources using CRUD ops and moving through states using hypermedia links

The Basics: Client-Server

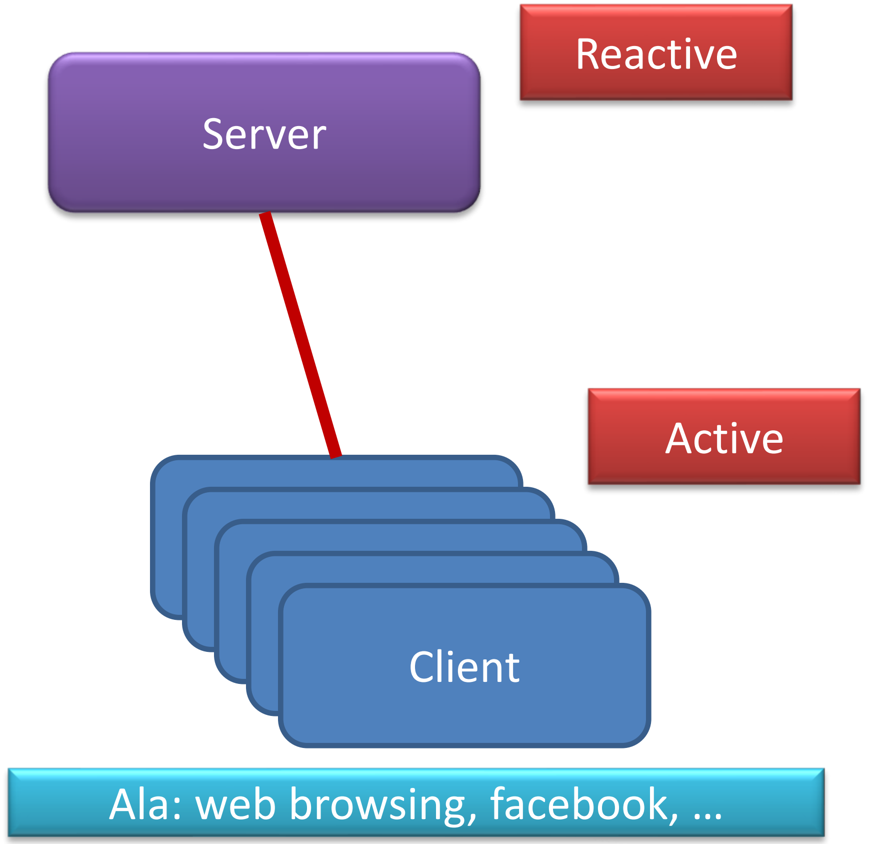
- Well defined roles of components and connectors…



Client-server architecture Two components need to communicate, and they are independent of each other, even running in different processes or being distributed in different machines. The two components are not equal peers communicating with each other, but one of them is initiating the communication, asking for a service that the other provides. Furthermore, multiple components might request the same service provided by a single component. Thus, the component providing a service must be able to cope with numerous requests at any time, i.e. the component must scale well. On the other hand, the requesting components using one and the same service might deal differently with the results. This asymmetry between the components should be reflected in the architecture for the optimization of quality attributes such as performance, shared use of resources, and memory consumption.

The CLIENT-SERVER pattern distinguishes two kinds of components: clients and servers. The client requests information or services from a server. To do so it needs to know how to access the server, that is, it requires an ID or an address of the server and of course the server's interface. The server responds to the requests of the client, and processes each client request on its own. It does not know about the ID or address of the client before the interaction takes place. Clients are optimized for their application task, whereas servers are optimized for serving multiple clients[3].

[3]Paris Avgeriou and Uwe Zdun, "Architectural patterns revisited – a pattern language", In 10th *European Conference on Pattern Languages of Programs* (EuroPlop), Irsee, 2005.



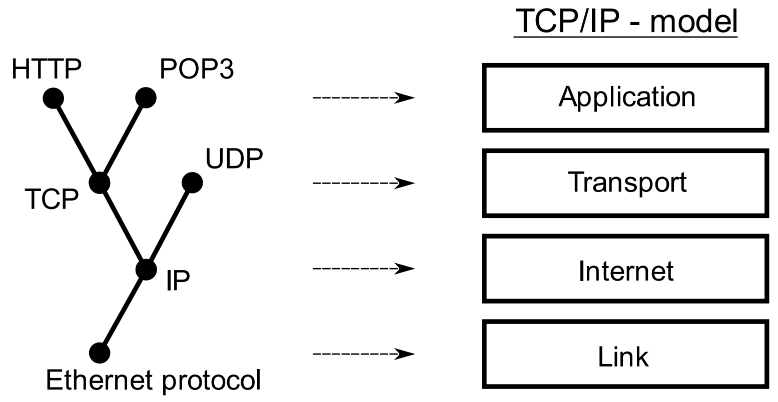
AARHUS UNIVERSITET

The Basic: 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



AARHUS UNIVERSITET

The Basis: HTTP

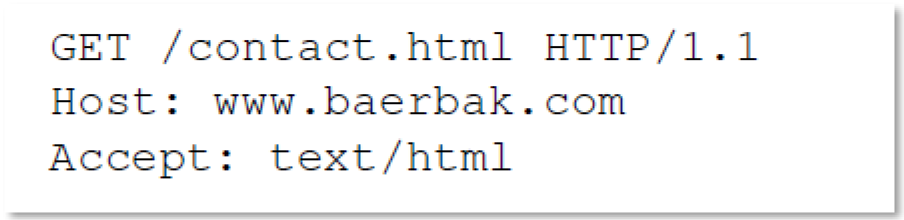
- HTTP = Hyper Text Transfer Protocol
  - Application Protocol for Distributed Information Systems
    - Exchanging information between clients and server
- Has four parts
  - Verbs: GET, POST, PUT, DELETE
    - Corresponds to normal database CRUD operations
  - Standardized data formats
    - Media types: text/html, image/gif, application/json
  - Message format in text
    - Verb + Headers (key/value) + empty line + body
  - Standard Error Code Vocabulary
    - 200 OK, 404 NOT FOUND, 201 CREATED, …

AARHUS UNIVERSITET

Message Format

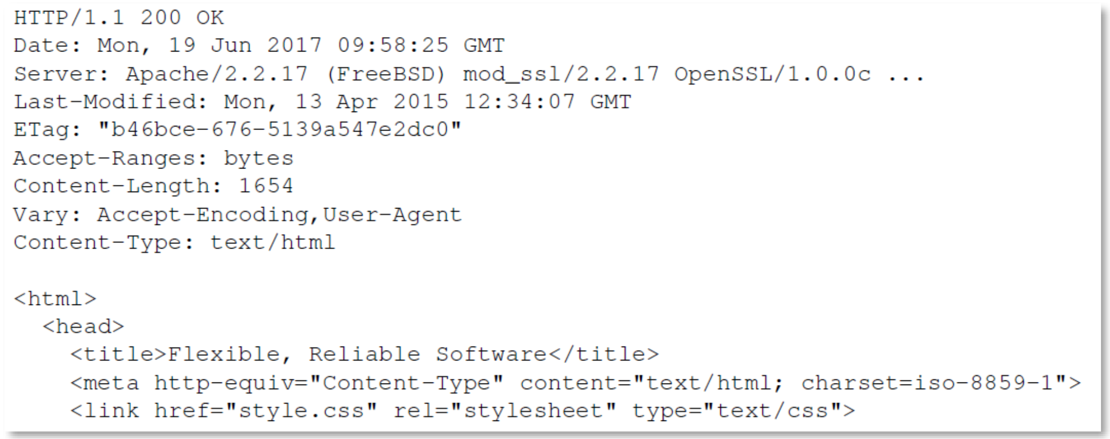
Text format !

- Request line
  - Verb resource HTTP version
  - Header key-values



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

- Reply line
  - Status line
    - HTTP codes
  - Header fields
  - Message body



```
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">
```

Roy Fielding's work

- *Goal: Keep the **scalable** hypermedia properties of WWW*
- REST = **RE**presentational **S**tate **T**ransfer
  - Transferring a *representation of data* in a format matching one of standard data types (media types)
  - *Resource*: any information that can be named
  - Identified by a *resource identifier*
    - *URI = Uniform Resource Identifier*
  - Interactions are *stateless*
    - Each request contains all the information necessary

Exercise: Why is everybody so keen on 'stateless'? What QA is involved?

AARHUS UNIVERSITET

Resource Identifier: URI

- 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://localhost:4567/bin**

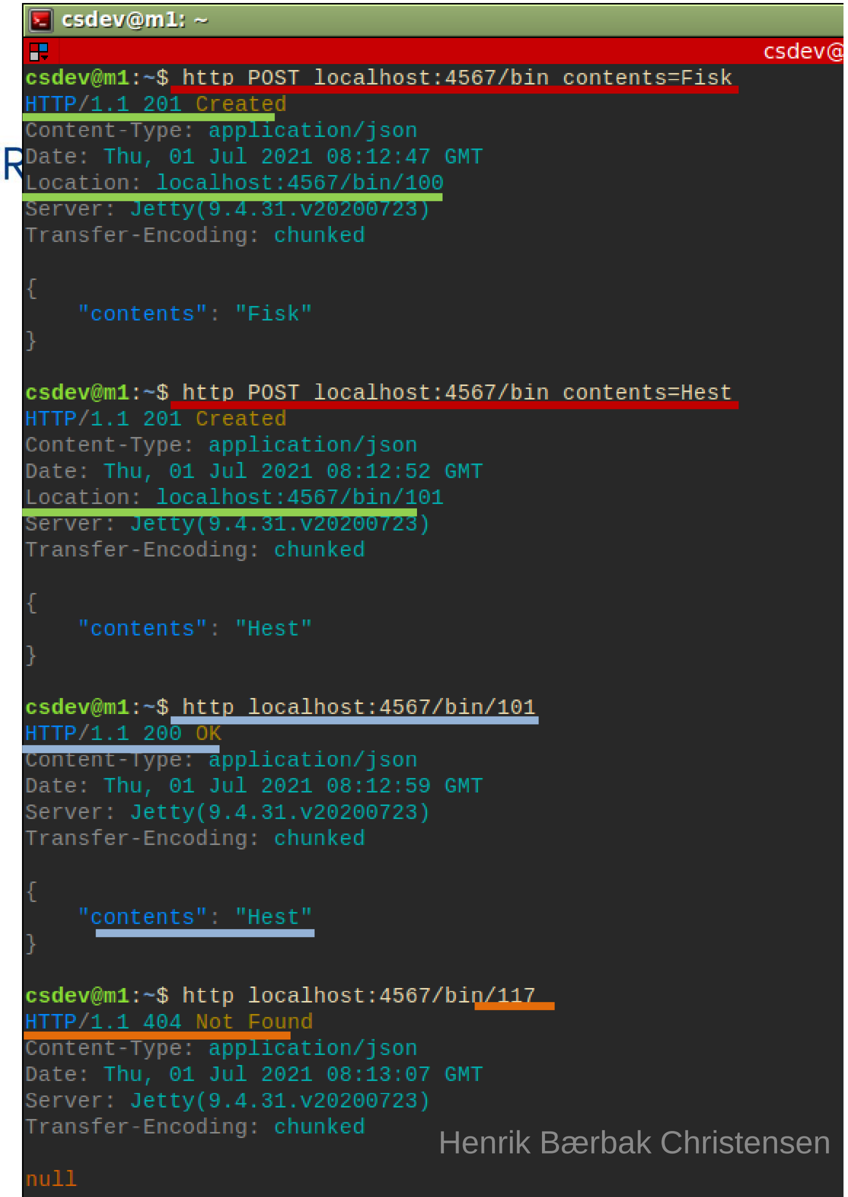
Exercise:
Identify the parts of the URI

AARHUS UNIVERSITET

CRUD by REST

- I can now design an *Information System* using the REST style. Example ‘a clipboard web server
  - POST on /pastebin/ with a message body
    - = CREATE a new clip (resource) on the clipboard, assign resource ID
  - GET on /pastebin/100
    - = READ the stored clip in the provided resource ID
  - PUT on /pastebin/100 with a complete new message body
    - = UPDATE the contents of the resource
  - DELETE on /pastebin/100
    - = you get it ☺

AARHUS UNIVER



```
csdev@m1: ~
csdev@m1:~$ http POST localhost:4567/bin contents=Fisk
HTTP/1.1 201 Created
Content-Type: application/json
Date: Thu, 01 Jul 2021 08:12:47 GMT
Location: localhost:4567/bin/100
Server: Jetty(9.4.31.v20200723)
Transfer-Encoding: chunked

{
    "contents": "Fisk"
}

csdev@m1:~$ http POST localhost:4567/bin contents=Hest
HTTP/1.1 201 Created
Content-Type: application/json
Date: Thu, 01 Jul 2021 08:12:52 GMT
Location: localhost:4567/bin/101
Server: Jetty(9.4.31.v20200723)
Transfer-Encoding: chunked

{
    "contents": "Hest"
}

csdev@m1:~$ http localhost:4567/bin/101
HTTP/1.1 200 OK
Content-Type: application/json
Date: Thu, 01 Jul 2021 08:12:59 GMT
Server: Jetty(9.4.31.v20200723)
Transfer-Encoding: chunked

{
    "contents": "Hest"
}

csdev@m1:~$ http localhost:4567/bin/117
HTTP/1.1 404 Not Found
Content-Type: application/json
Date: Thu, 01 Jul 2021 08:13:07 GMT
Server: Jetty(9.4.31.v20200723)
Transfer-Encoding: chunked

null
```

Demo

- POST ‘Fisk and ‘Hest’ in bins

- Assigned bin 100, 101

- GET bin 101
- Which is ‘Hest’

- GET bin 117
- Which is not found (404)

AARHUS UNIVERSITET

HATEOAS

- One drawback of REST compared to other programming models
  - In oo/procedural/functional you can define *methods* that do complex algoritms over multiple objects/"resources"
    - Not just: create, read, update, delete

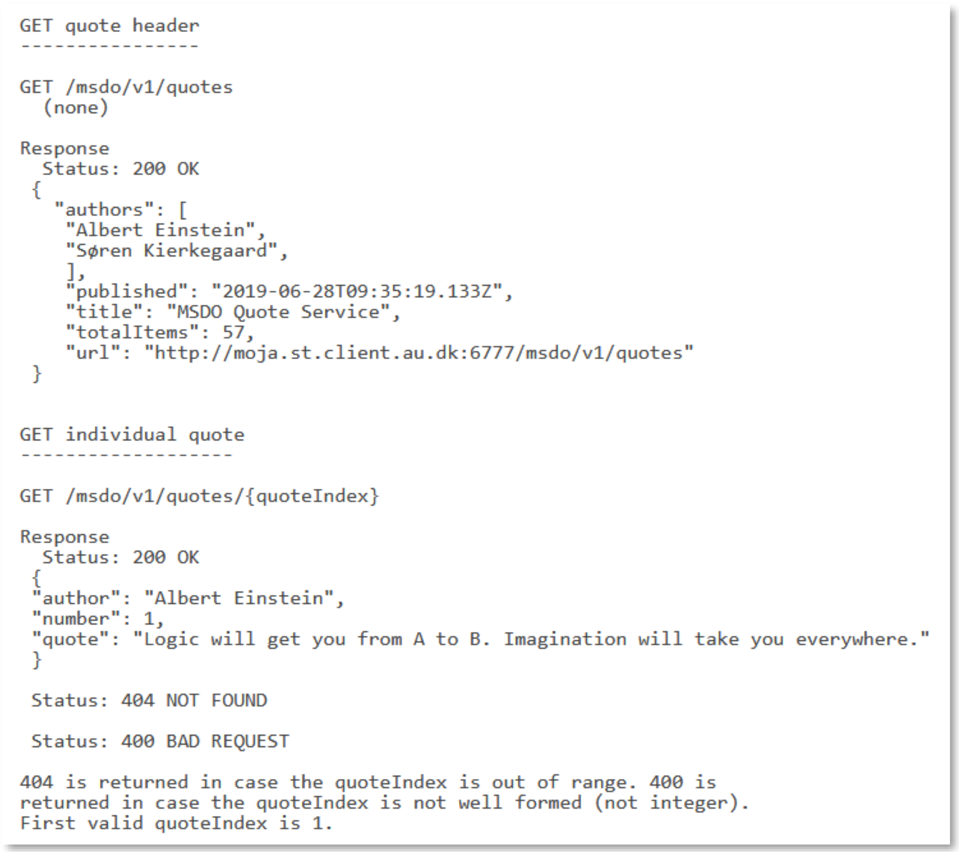
- Solution: Hyper Text As The Engine Of Application State
  - Any resource contains not just its *state* but also *links that may modify state of related resources*

  - **Read FRS §7.** HATEOAS is beyond our MSDO scope…

AARHUS UNIVERSITET

Define the API

- FRDS §7.7 presents a rough template for API definition
- Example
- Will be used in MSDO
- Or use swagger or …



```
GET quote header
----------------

GET /msdo/v1/quotes
  (none)

Response
  Status: 200 OK
 {
   "authors": [
    "Albert Einstein",
    "Søren Kierkegaard",
    ],
    "published": "2019-06-28T09:35:19.133Z",
    "title": "MSDO Quote Service",
    "totalItems": 57,
    "url": "http://moja.st.client.au.dk:6777/msdo/v1/quotes"
 }


GET individual quote
-------------------

GET /msdo/v1/quotes/{quoteIndex}

Response
  Status: 200 OK
 {
 "author": "Albert Einstein",
 "number": 1,
 "quote": "Logic will get you from A to B. Imagination will take you everywhere."
 }

 Status: 404 NOT FOUND

 Status: 400 BAD REQUEST

404 is returned in case the quoteIndex is out of range. 400 is
returned in case the quoteIndex is not well formed (not integer).
First valid quoteIndex is 1.
```

The Slow Version

HTTP & ReST

- The Broker pattern had its glory in the early 1990’ies as a paradigm for distributed communication

- However, the WWW sort of happened in the same period.

- And soon it was realized that HTTP could do *much more* than just provide web pages…

AARHUS UNIVERSITET

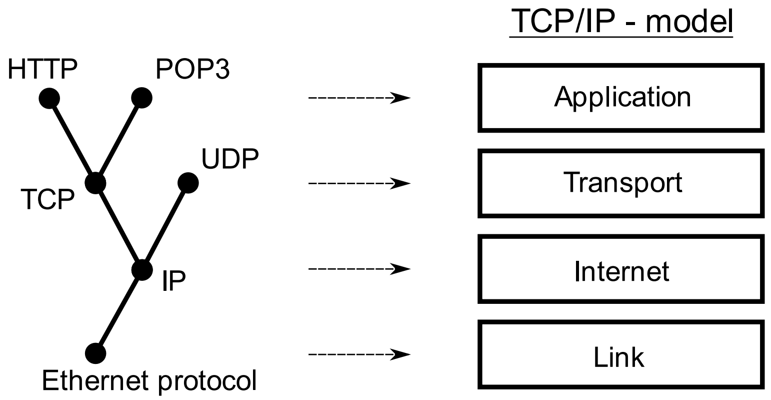
HTTP

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



AARHUS UNIVERSITET

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 ther documents.
  - HTTP: Hypertext Transfer Protocol is an **application protocol** for distributed information systems.
  - WWW: The **system** made that used HTML+HTTP to share documents at CERN, and later – quite a few other places ☺

AARHUS UNIVERSITET

Message Format

Text format !

- Request line
  - Verb resource HTTP version
  - Header key-values

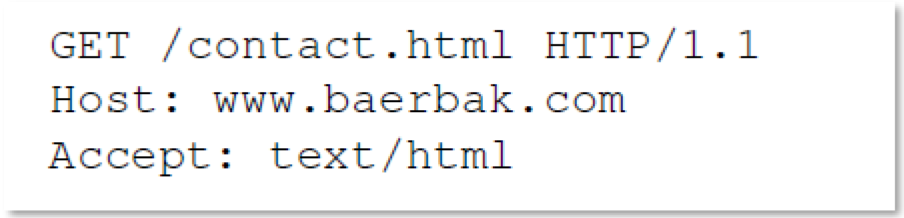
```
GET /contact.html HTTP/1.1
Host: www.baerbak.com
Accept: text/html
```

- Reply line
  - Status line
    - HTTP codes
  - Header fields
  - Message body

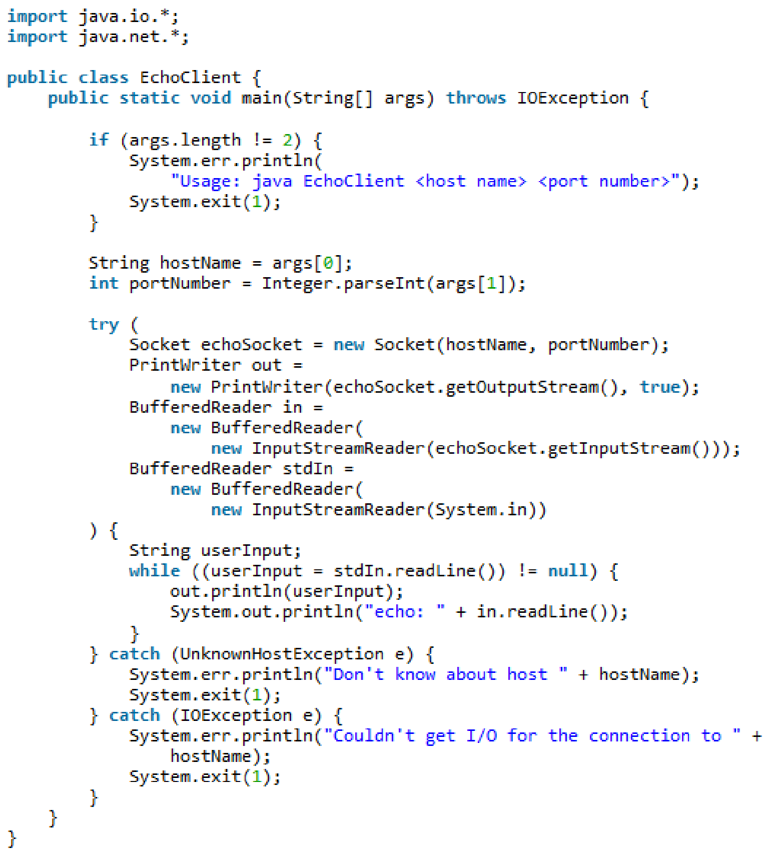
```
HTTP/1.1 200 OK
Date: Mon, 19 Jun 2017 09:58:25 GMT
Server: Apache/2.2.17 (FreeBSD) mod_ssl/2.2.17 OpenSSL/1.0.0c ...
Last-Modified: Mon, 13 Apr 2015 12:34:07 GMT
ETag: "b46bce-676-5139a547e2dc0"
Accept-Ranges: bytes
Content-Length: 1654
Vary: Accept-Encoding,User-Agent
Content-Type: text/html

<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



```
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);
        }
    }
}
```

URI / URL

- URI: Uniform Resource Identifier

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

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

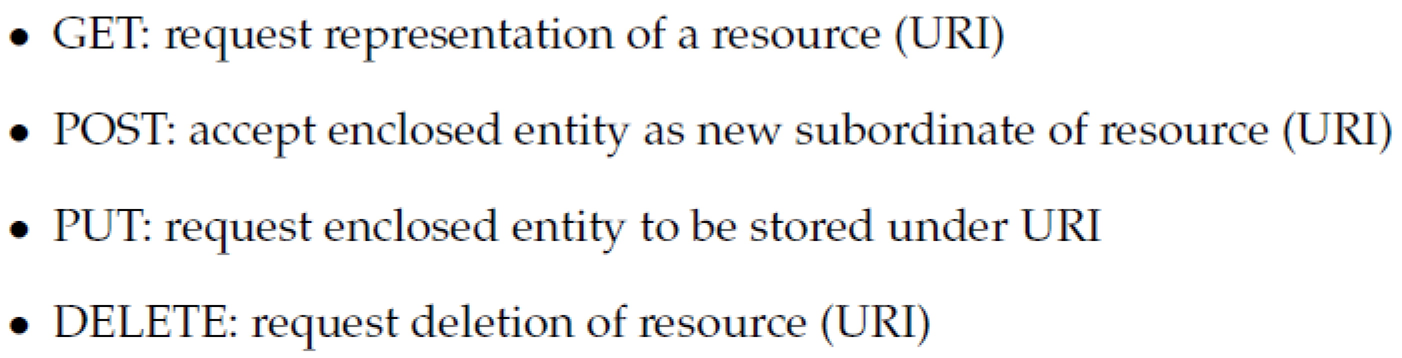
- URL = URI in which resource location and means are defined
  - **http**://www.baerbak.com/contact.html
  - **http://localhost:4567/bin**

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…***

AARHUS UNIVERSITET

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!**

AARHUS UNIVERSITET

POST

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

- Consider ‘telemed.processAndStore(obs);’
  - Command pattern: *Convert method call to an object*
- *Now, consider that ‘telemed is on the server side*
  - POST allows us to ***create a command object***
    - POST /telemed HTTP/1.1
    - Body { method: ‘processAndStore’, argument: {‘sys’:140, …} }

PUT, DELETE

- PUT means ‘update’
  - That is, given an *existing* resource, overwrite its information with updated information (*)
  - **Mutator ☺**

- DELETE means ‘delete’ ☺
  - That is, remove an existing resource from the server

- (*) unless you only provide a *partial ressource*, then you should update using POST instead (which IMO does not make sense, but…)

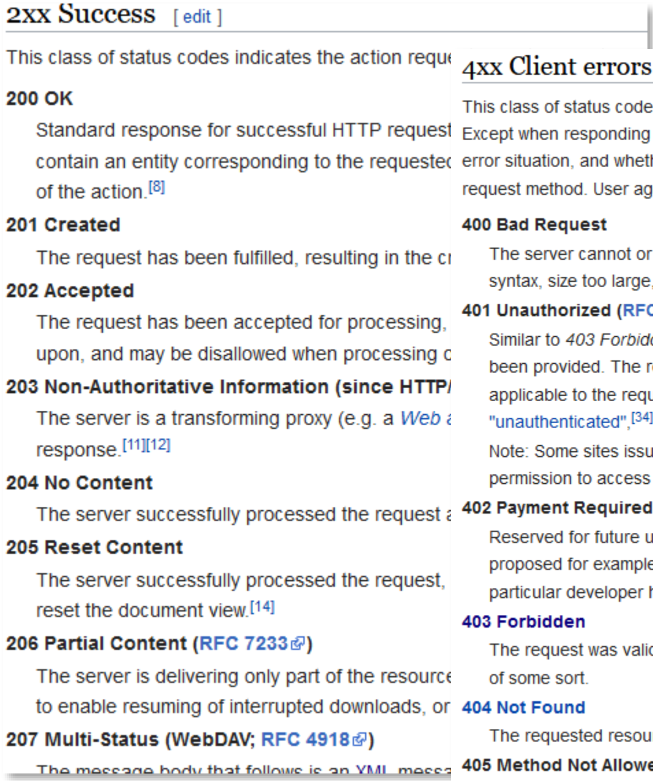
Failures in Distribution

- 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 requestec
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 c

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

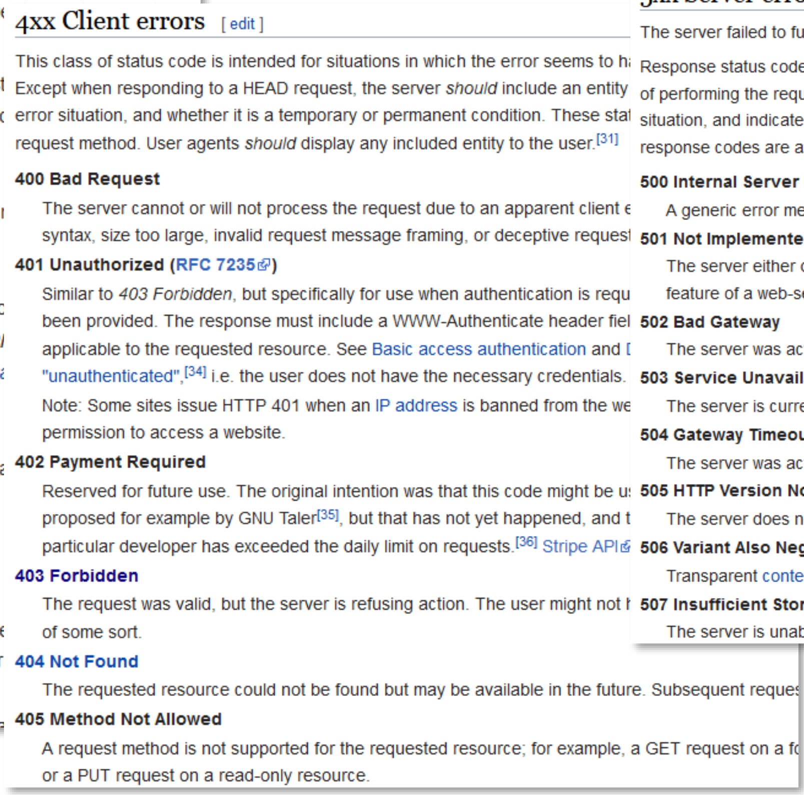
The server successfully processed the request,
reset the document view.[14]

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 stat
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

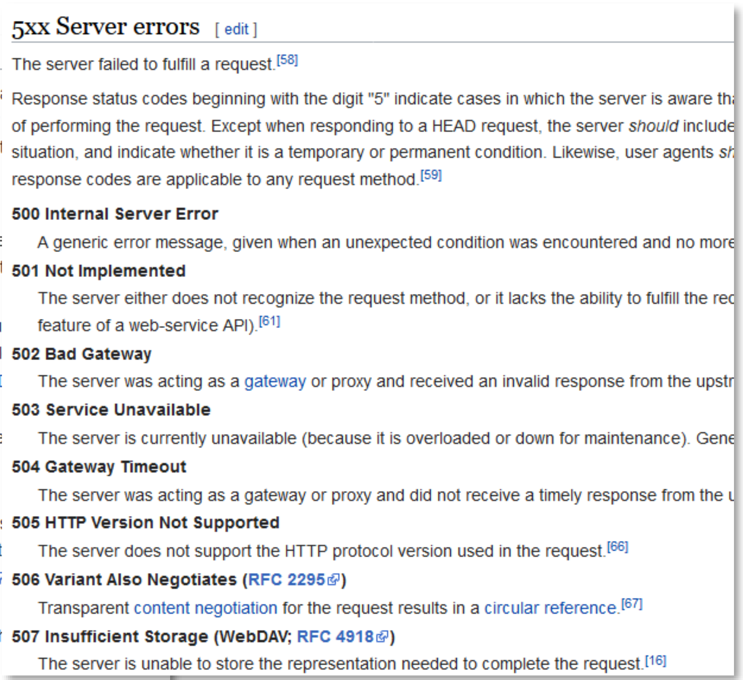
The request was valid, but the server is refusing action. The user might not h
of some sort.

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]

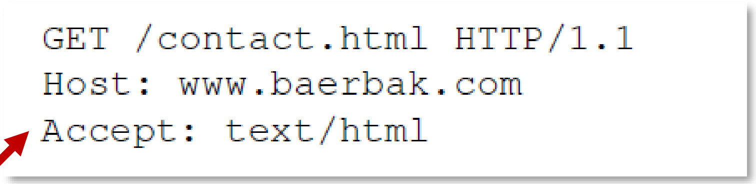
507 Insufficient Storage (WebDAV; RFC 4918)

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

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

AARHUS UNIVERSITET

REpresentation State Transfer

What is REST

- As a software architect, I see it as an
  - **Architectural style / pattern**
- It is simply quite another programming model
  - Functional programming:
    - Computation is passing data through chains of functions
  - Object programming:
    - Computation is community of objects passing messages
  - RPC over Client-Server:
    - Computation is clients invoking procedures on remote servers
  - REST
    - Computation is clients manipulating resources using CRUD ops and moving through states using hypermedia links

Programming Model

- Broker pattern
  - Supports RPC/RMI between clients and servers
    - State changes through accessors and mutator methods
    - Any interface is possible

- REST
  - Supports only CRUD on remote resources (=Data objects)
  - Supports workflow through hypermedia links

- **Very different programming model required compared to RPC**
- **Not all architectures are suited for REST !**

Roy Fielding's work

- *Goal: Keep the* ***scalable*** *hypermedia properties of WWW*
- REST = **RE**presentational **S**tate **T**ransfer
  - Transferring a *representation of data* in a format matching one of standard data types (media types)
  - *Resource*: any information that can be named
  - Identified by a *resource identifier*
    - *URI = Uniform Resource Identifier*
  - Interactions are *stateless*
    - Each request contains all the information necessary

Exercise: Why is everybody so keen on 'stateless'? What QA is involved?

AARHUS UNIVERSITET

Representing Resources

Using TeleMed as case

AARHUS UNIVERSITET

Example

- Resource: Inger’s blood pressure measured on 29/6/2017
- Representation of data using standard media type:
  - { pid: ”251248-1234”, sys: 120.0, dia:70.0 }   (json)
- Resource identifier
  - http://telemed.org/bp/251248-1234/made-29-06-2017-09-59-17
  - I.e. Inger’s resource (her blood pressure measurement) is uniquely identified using this URI

AARHUS UNIVERSITET

Example: CRUD

- Inger makes the measurement CREATE
- POST /bp
  - Body: { pid: ”251248-1234”, sys: 120.0, dia:70.0 }
- Response
  - StatusCode: 201 CREATED
  - Location: /bp/251248-1234/made-29-06-2017-09-59-17
  - Body: { pid: ”251248-1234”, sys: 120.0, dia:70.0, status: ”new” }

- Meaning
  - The resources was created, has resource id
    - /bp/251248-1234/made-29-06-2017-09-59-17

AARHUS UNIVERSITET

Example: CRUD

- Inger reviews the measurement READ
- GET /bp/251248-1234/made-29-06-2017-09-59-17
  - Body: (none)
- Response
  - StatusCode: 200 OK
  - Body: { pid: ”251248-1234”, sys: 120.0, dia:70.0, status=”new” }
- Meaning
  - The resources was found, and the measurement returned

AARHUS UNIVERSITET

Example: CRUD

- Inger updates the measurement UPDATE
- PUT /bp/251248-1234/made-29-06-2017-09-59-17
  - Body: { pid: ”251248-1234”, sys: 126.0, dia:69.0 }
- Response
  - StatusCode: 201 CREATED
  - Body: { pid: ”251248-1234”, sys: 126.0, dia:69.0, status=”revised” }
- Meaning
  - The resources was found, and the measurement updated

AARHUS UNIVERSITET

Example: CRUD

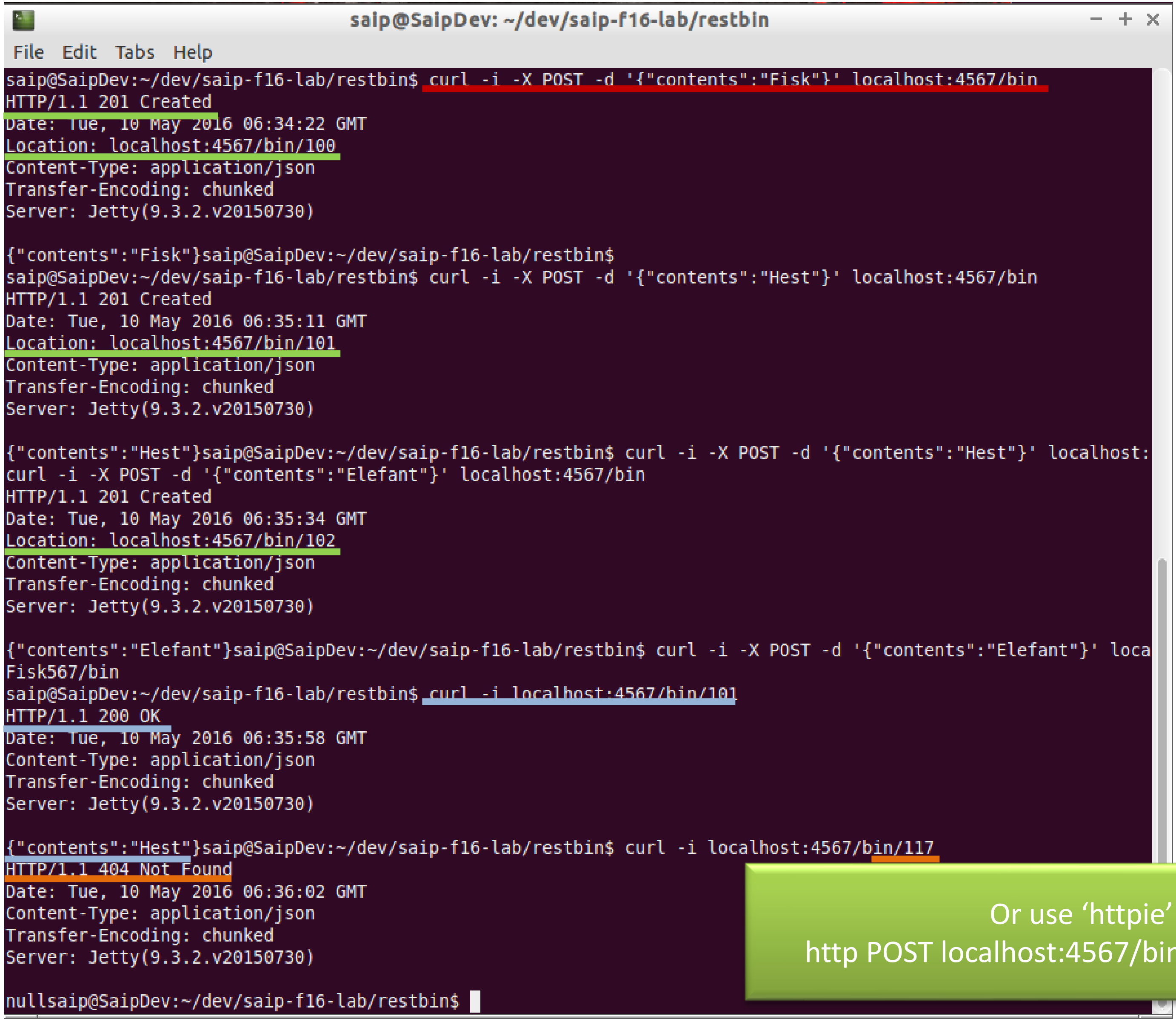
- Inger deletes the measurement DELETE
- DELETE /bp/251248-1234/made-29-06-2017-09-59-17
  - Body: (none)
- Response
  - StatusCode: 204 No Content
  - Body: none
- Meaning
  - The resources was found, and the measurement deleted

Prototype: pastebin

- REST is pretty lightweight programming wise…
  - Goal: AP to demonstrate ”pastebin”
    - Online service for storing text messages = ‘post-its’
  - Total time: 1.5 hour (well – a bit cheating)

- Developed
  - Webserver, accepting POST and GET
    - Using Spark-java framework (IPC) and GSON (Marshaling)
  - Client: curl or httpie ☺

Demo



```
saip@SaipDev:~/dev/saip-f16-lab/restbin$ curl -i -X POST -d '{"contents":"Fisk"}' localhost:4567/bin
HTTP/1.1 201 Created
Date: Tue, 10 May 2016 06:34:22 GMT
Location: localhost:4567/bin/100
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.3.2.v20150730)

{"contents":"Fisk"}saip@SaipDev:~/dev/saip-f16-lab/restbin$
saip@SaipDev:~/dev/saip-f16-lab/restbin$ curl -i -X POST -d '{"contents":"Hest"}' localhost:4567/bin
HTTP/1.1 201 Created
Date: Tue, 10 May 2016 06:35:11 GMT
Location: localhost:4567/bin/101
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.3.2.v20150730)

{"contents":"Hest"}saip@SaipDev:~/dev/saip-f16-lab/restbin$ curl -i -X POST -d '{"contents":"Hest"}' localhost:
curl -i -X POST -d '{"contents":"Elefant"}' localhost:4567/bin
HTTP/1.1 201 Created
Date: Tue, 10 May 2016 06:35:34 GMT
Location: localhost:4567/bin/102
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.3.2.v20150730)

{"contents":"Elefant"}saip@SaipDev:~/dev/saip-f16-lab/restbin$ curl -i -X POST -d '{"contents":"Elefant"}' loca
Fisk567/bin
saip@SaipDev:~/dev/saip-f16-lab/restbin$ curl -i localhost:4567/bin/101
HTTP/1.1 200 OK
Date: Tue, 10 May 2016 06:35:58 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.3.2.v20150730)

{"contents":"Hest"}saip@SaipDev:~/dev/saip-f16-lab/restbin$ curl -i localhost:4567/bin/117
HTTP/1.1 404 Not Found
Date: Tue, 10 May 2016 06:36:02 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.3.2.v20150730)

nullsaip@SaipDev:~/dev/saip-f16-lab/restbin$
```

- POST ‘Fisk’, ‘Hest’ and ‘Elefant’ in bins
- Assigned bin 100, 101, 102

- GET bin 101
- Which is ‘Hest’

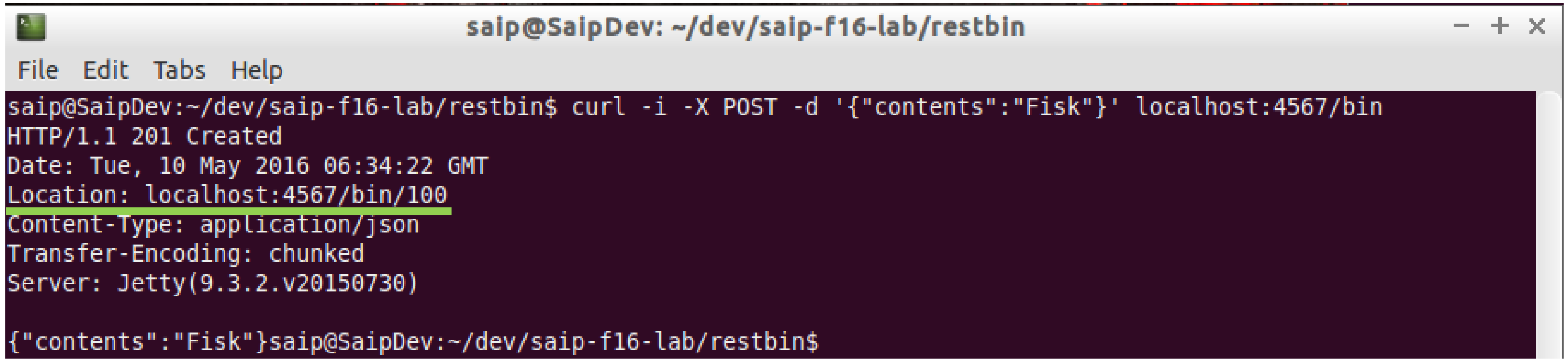
- GET bin 117
- Which is not found (404)

Or use ‘httpie’:
http POST localhost:4567/bin contents=Fisk

AARHUS UNIVERSITET

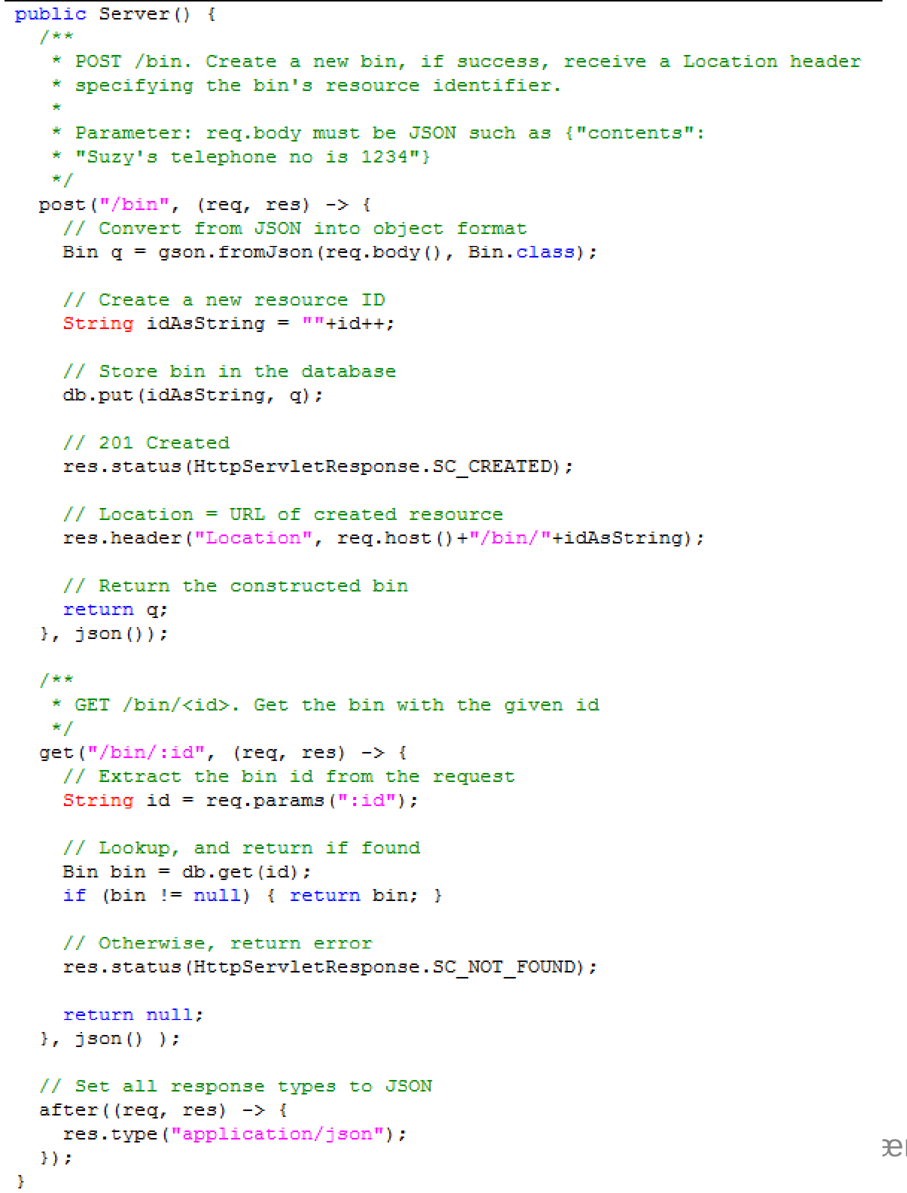
Note

- POST of course needs to tell client the *resource identifier* of the newly created object!
  - Reponse contains ‘Location’ field



saip@SaipDev: ~/dev/saip-f16-lab/restbin

```
saip@SaipDev:~/dev/saip-f16-lab/restbin$ curl -i -X POST -d '{"contents":"Fisk"}' localhost:4567/bin
HTTP/1.1 201 Created
Date: Tue, 10 May 2016 06:34:22 GMT
Location: localhost:4567/bin/100
Content-Type: application/json
Transfer-Encoding: chunked
Server: Jetty(9.3.2.v20150730)

{"contents":"Fisk"}saip@SaipDev:~/dev/saip-f16-lab/restbin$
```



```
public Server() {
  /**
   * POST /bin. Create a new bin, if success, receive a Location header
   * specifying the bin's resource identifier.
   *
   * Parameter: req.body must be JSON such as {"contents":
   * "Suzy's telephone no is 1234"}
   */
  post("/bin", (req, res) -> {
    // Convert from JSON into object format
    Bin q = gson.fromJson(req.body(), Bin.class);

    // Create a new resource ID
    String idAsString = ""+id++;

    // Store bin in the database
    db.put(idAsString, q);

    // 201 Created
    res.status(HttpServletResponse.SC_CREATED);

    // Location = URL of created resource
    res.header("Location", req.host()+"/bin/"+idAsString);

    // Return the constructed bin
    return q;
  }, json());

  /**
   * GET /bin/<id>. Get the bin with the given id
   */
  get("/bin/:id", (req, res) -> {
    // Extract the bin id from the request
    String id = req.params(":id");

    // Lookup, and return if found
    Bin bin = db.get(id);
    if (bin != null) { return bin; }

    // Otherwise, return error
    res.status(HttpServletResponse.SC_NOT_FOUND);

    return null;
  }, json() );

  // Set all response types to JSON
  after((req, res) -> {
    res.type("application/json");
  });
}
```

Server code

- A PasteBin server in 50 lines of Java
  - OK, Spark-java helps quite a bit!

Is in the 'FRDS.Broker' codebase.

Left as an Exercise

- We should be able to *update* a text in pastebin
  - PUT verb

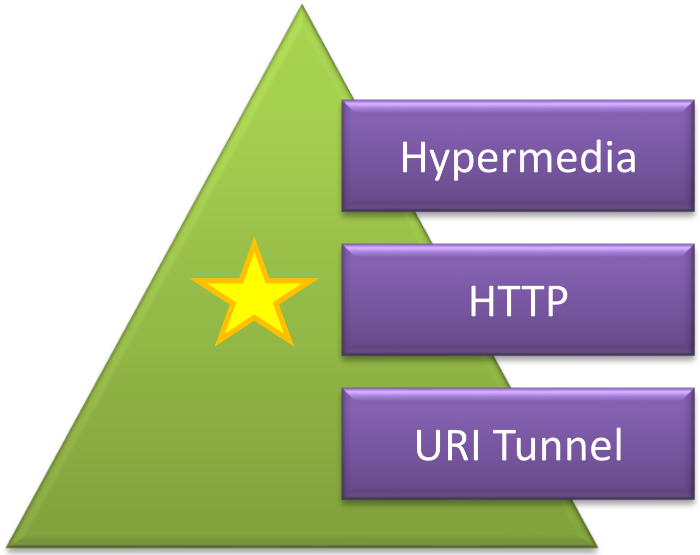
- And delete an entry
  - DELETE verb

Discussion

- REST uses the **HTTP as designed**
  - CRUD verbs and Status Codes (methods, return type)
    - Virtually allows all *Information Systems* operations !
  - URLs as resource identifiers (location+object)
    - Always identify the *same* resource, and representation of state is *always communicated*
  - Well defined *data representations* (media types)
    - JSON has become favorite (readable + small footprint)

Richardson’s Maturity model

- From low maturity to high maturity
  - URI Tunnel
    - Just use HTTP as IPC layer
      - SOAP, WSDL, WebServices
      - And our URI Tunnel Broker!
  - HTTP
    - Use CRUD Verbs on resources
  - Hypermedia
    - Use links to define workflows



AARHUS UNIVERSITET

Level 2 REST

AARHUS UNIVERSITET

Workflow

- Business systems can often be modelled as workflows
  - CS term: State machines / state graphs ☺

- Ex:Book a flight

– I *search* for flights available	get list of links
– I pick one particular flight	get ‘book’ link
– I *book* the flight	enter personal details
– I *pay* for the flight	enter credit card details
– I *get* a) e-ticket b) receipt	get two links

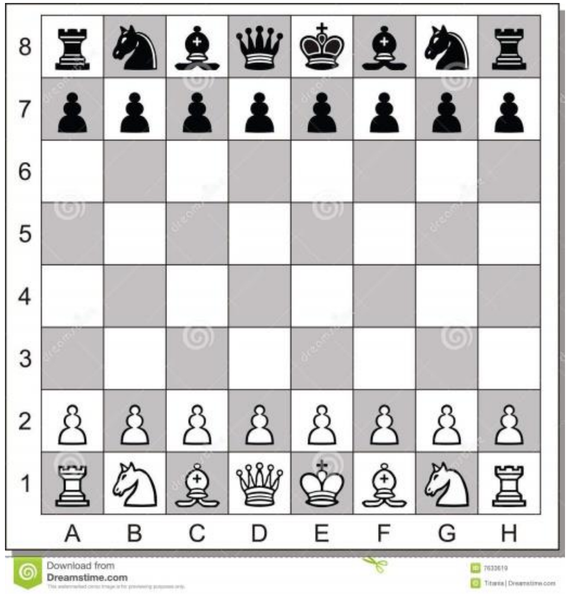
Exercise

- I *search* for flights
  - What HTTP verb is that? What resources are involved?
- I *book* the flight
  - What HTTP verb is that? What resources are involved?
- I *pay* for the flight
  - What HTTP verb is that? What resources are involved?
- I *get* my e-ticket
  - What HTTP verb is that? What resources are involved?

AARHUS UNIVERSITET

Level 2: Hypermedia

- Workflows are not just ‘CRUD a resource’, rather more complex
  - Transactions: Multiple entities atomically updated
  - State transitions: *Mutator* methods that updates several entities and/or updates state
  - Ex: A game’s move(f,t) method
    - Validate move (may return ‘not valid’)
    - Update board state (transaction, e.g. king castling)



Analysis

- ‘move()’ using HTTP verbs ???

- Analysis A:
  - ”No can do”
    - Because ‘move’ is not a create, it is not a read, nor update, nor delete of a single resource (stateless)

AARHUS UNIVERSITET

Analysis

- ‘move()’ using HTTP verbs

- Analysis B: *Maybe it is an update of game*
  - PUT /game/47
  - Body: Full board state with the move executed
    - But – then the server has to *infer* the move from the *delta between state ‘before’ and state ‘after’* which is weird!
      - And it is definitely not **stateless** – right?

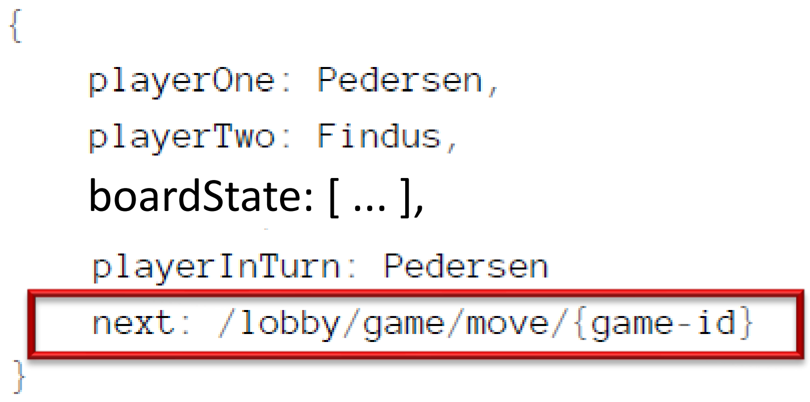
AARHUS UNIVERSITET

Analysis

- Analysis C: *A 'state transition resource'*
  - *Creating a game, is creation of **two** resources*
    - The game resource /game/47/
    - The **move** resource /game/47/move or /game/move/47
  - PUT /game/47/move
  - Body: { from: e2, to: e4, player:white}
- This will
  - Try to UPDATE the state => 200 OK or 401 Invalid
  - If 200 OK, then the game resource is updated
    - And can be successively GET to see new board state

Challenge

- But how do we return **two** resources from the game create POST message?
  - We can not, but we can use the WWW way – provide hypermedia links!!!



```
{
    playerOne: Pedersen,
    playerTwo: Findus,
    boardState: [ ... ],
    playerInTurn: Pedersen
    next: /lobby/game/move/{game-id}
}
```

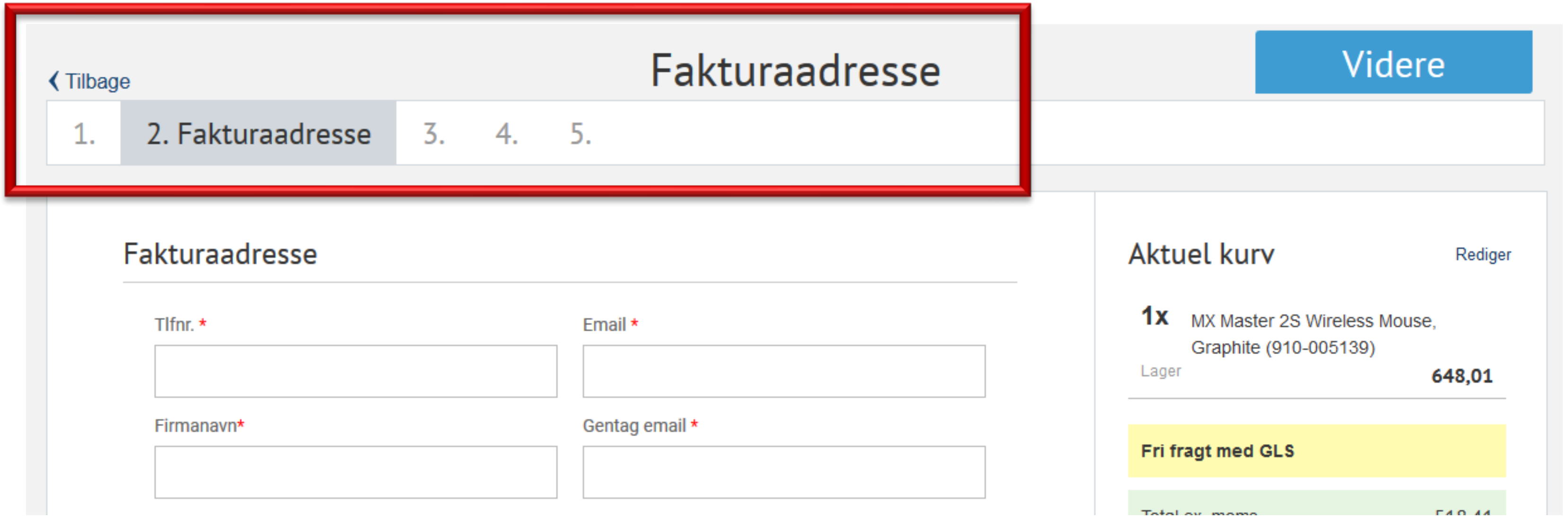
Aka

- HATEOAS:
  - *Hypermedia As The Engine Of Application State.*

- Application state changes are modelled as hypermedia links, each to a resource that objectify the change itself, not the old/new state of underlying objects
  - A ‘move’ resource, a ‘payment’ resource, a ‘send items to address’ resource, etc.

Often visible in UI

- The state changes of the *order*



❮ Tilbage

Fakturaadresse

Videre

1. 2. Fakturaadresse 3. 4. 5.

Fakturaadresse

Tlfnr. *

Email *

Firmanavn*

Gentag email *

Aktuel kurv

Rediger

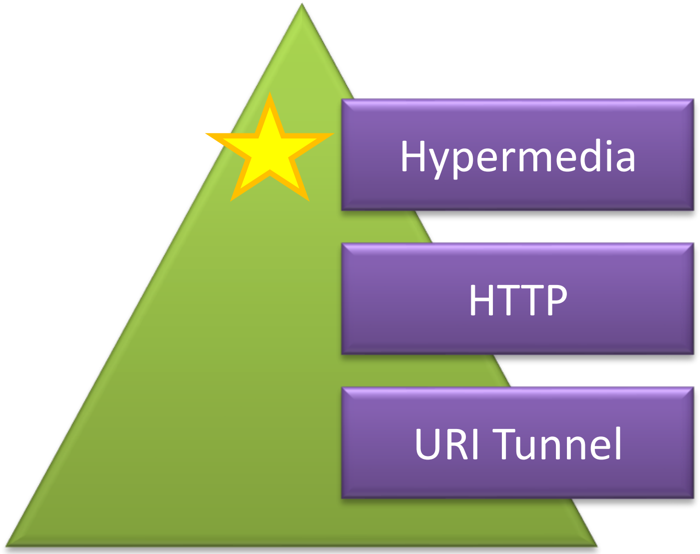
1x MX Master 2S Wireless Mouse, Graphite (910-005139)

Lager 648,01

Fri fragt med GLS

Level 2: Hypermedia

- So – REST is a radically different architectural pattern/style, different from OO and interface-based paradigms
- POST to create a resource
  - May return several hypermedia links that define valid state transitions for the resource
    - Which are then manipulated through the HTTP verbs
  - Makes potential state transitions *discoverable*
    - Just like any new web page presents links that I may follow



AARHUS UNIVERSITET

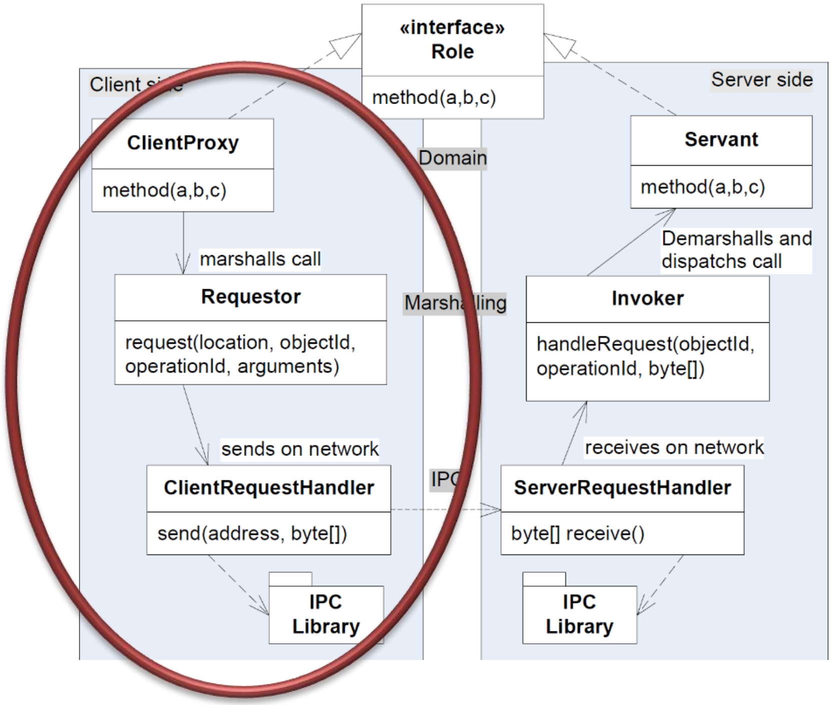
REST versus Broker

Comparing Apples and Bananas?

Programming Model

- Broker: invoke methods on objects
- REST: CRUD methods on resources

- Comparing to Broker
  - REST actually addresses responsibilities on both the Marshalling, Location, and IPC level.
  - *REST has much lower cohesion and tighter binding!*



AARHUS UNIVERSITET

Programming Model

- Broker is well supported by tooling
  - Java RMI, .Net remoting, …
- REST is (IMO – or am I missing something) more up to you to code it all
  - Swagger a.o. can provide templating
- REST is much tighter coupled to the HTTP platform
  - But it is a strong scaleable one, so …

Example

- Requirement
  - Rewrite the ‘cmd-daemon’ protocol to use RabbitMQ message broker

- Using FRDS.Broker
  - A task that takes about 1-2 hours
    - Using the RPC tutorial of RabbitMQ
- Using REST
  - Rewrite *everything from scratch*

Summary

- UR Tunnelling
  - Just uses HTTP and web technology/frameworks as the IPC layer in the Broker
    - That is : transport network packages to/from client and server

- REST
  - Architectural Pattern what deeply exploits HTTPs advantages
  - Lightweight with less tool support
  - Focus is on performance and scalability because
    - True Client-server No callback/observer pattern
    - Value passing of information

Summary

- Broker pattern and REST?

- **REST and OO are two different architectural styles…**