REpresentational State Transfer (REST)

Acknowledgements

This lecture has been inspired by and based off the work of NetApps GTAs from the past:

Kelvin Aviles	Fall 2017, Spring 2018
Prakriti Gupta	Fall 2016, Spring 2017
Gaurang Naik	Fall 2015
Thaddeus Czauski	Fall 2014

Outline

1. What is REST?
2. What is HTTP?
    a. Requests and Methods
    b. Response and Status Codes
    c. Examples
3. Multipart Request/Response
4. Caching
5. Cookies
6. REST Properties

What is REST?

REST

- Based off of Hypertext Transfer Protocol (HTTP).
- Often incorrectly used interchangeably with HTTP.
- Has sadly become somewhat of a buzzword as a result.
- NOT a protocol like HTTP.
- It describes how a protocol should be used, like a set of principles or rules.
- Can be used with protocols besides HTTP but is mostly only used for HTTP.

Dr. Roy Fielding

- Described REST as part of his dissertation in 2000 at UC Irvine.
- Did this while working on HTTP 1.1
- Co-Founded Apache Server
- HTTP Work probably contributed to confusion with REST.

We will come back to REST...

What is HTTP?

HTTP

RFC2616:

"The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers."

HTTP Request/Response Diagram

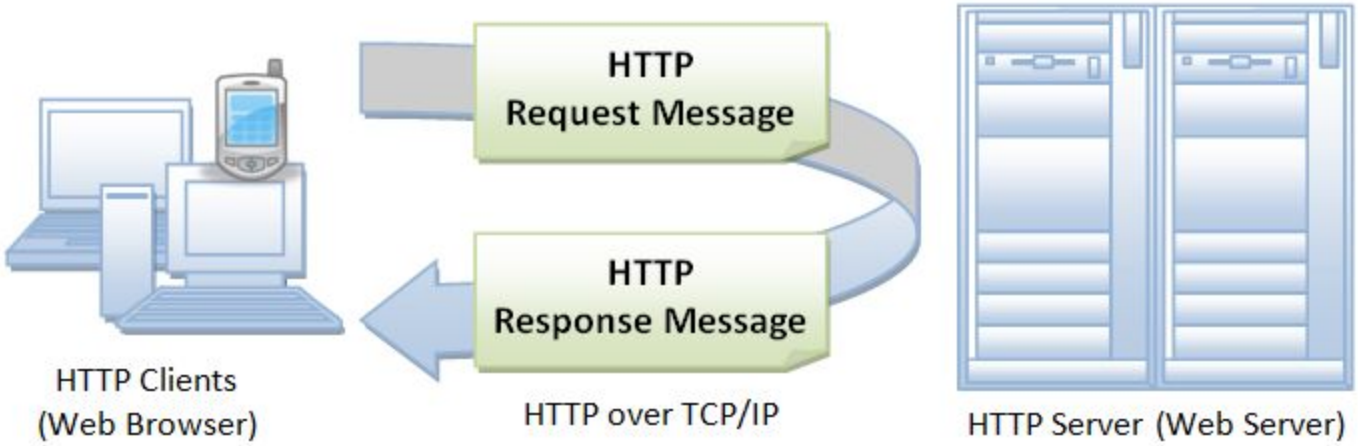
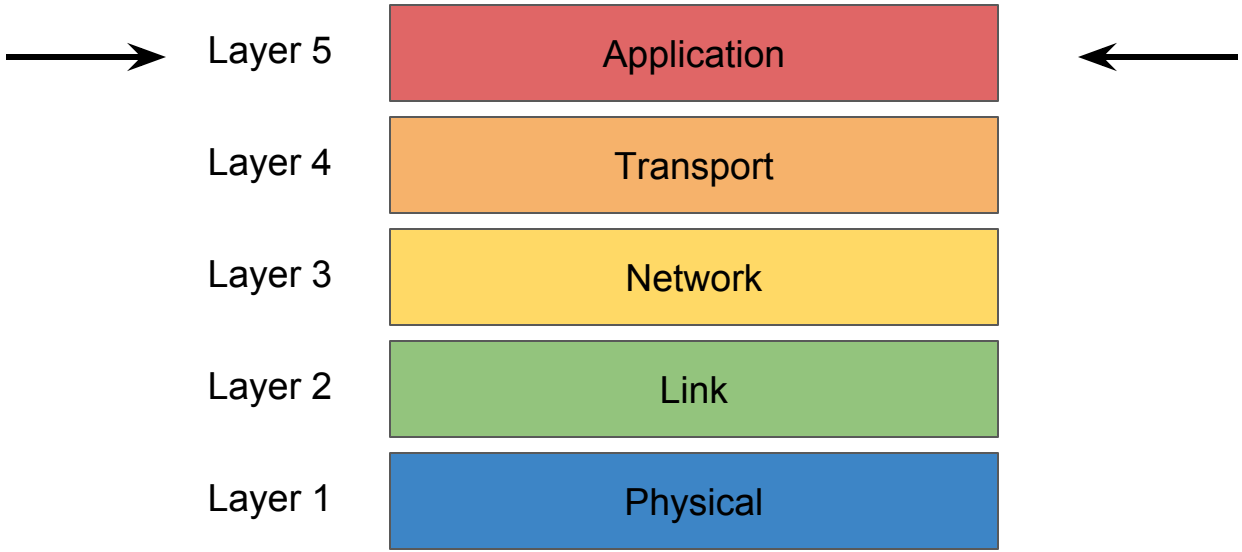


Image Ref: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Internet Layers



HTTP Requests and Responses

- google.com
- twitter.com
- wolframalpha.com
- vt.edu
- Each are part of an HTTP request.

HTTP Request Format

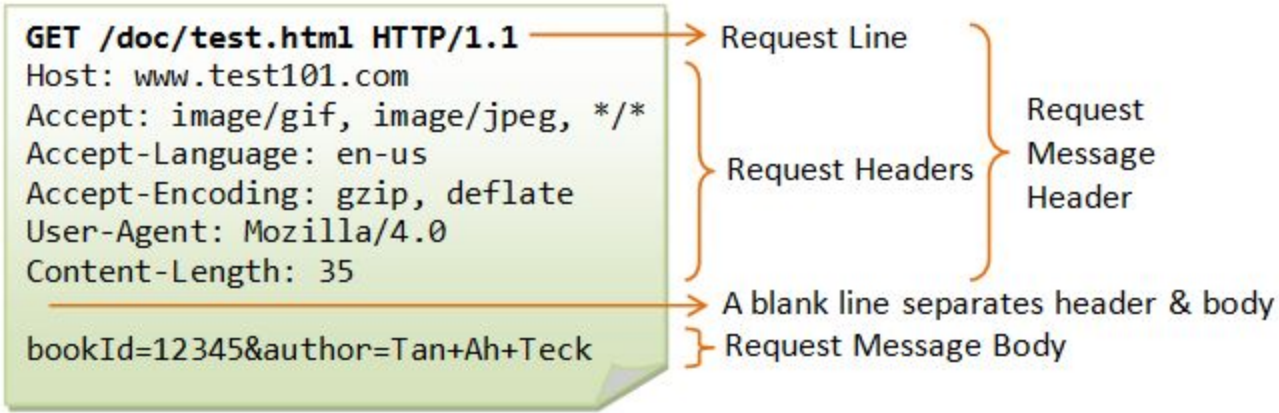


Image Ref: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Which would have been typed in URL bar?

A. http://www.test101.com/doc/test.html?bookId=12345&author=Tan+Ah+Teck
B. http://www.test101.com
C. http://www.test101.com/doc/test.html
D. None of the above

HTTP Request Line

request-line = method SP request-target SP HTTP-version CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

HTTP Request Line

request-line = method SP request-target SP HTTP-version CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

HTTP Request Line

request-line = method SP request-target SP HTTP-version CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

HTTP Request Line

request-line = method SP request-target SP HTTP-version CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

CRLF?

- CRLF is how the Server knows the request line has ended.
- What comes next?
  - Request Headers
  - Request Body

HTTP Methods

- **GET** – request (or "get") for a piece of resource from a HTTP server.
- **POST** – used to "post" additional data up to the server (e.g., submitting HTML form data or uploading a file).
- **HEAD** – request only the response header.
- **PUT** – used to update a file/resource on the server.
- **DELETE** – delete a resource present on the server.
- **OPTIONS** – query the server for a list of supported HTTP methods.
- **CONNECT** – creates TCP/IP connection.
- **TRACE** – echo the received request so client knows changes that occurred on server.

Note on REST and HTTP Request Methods

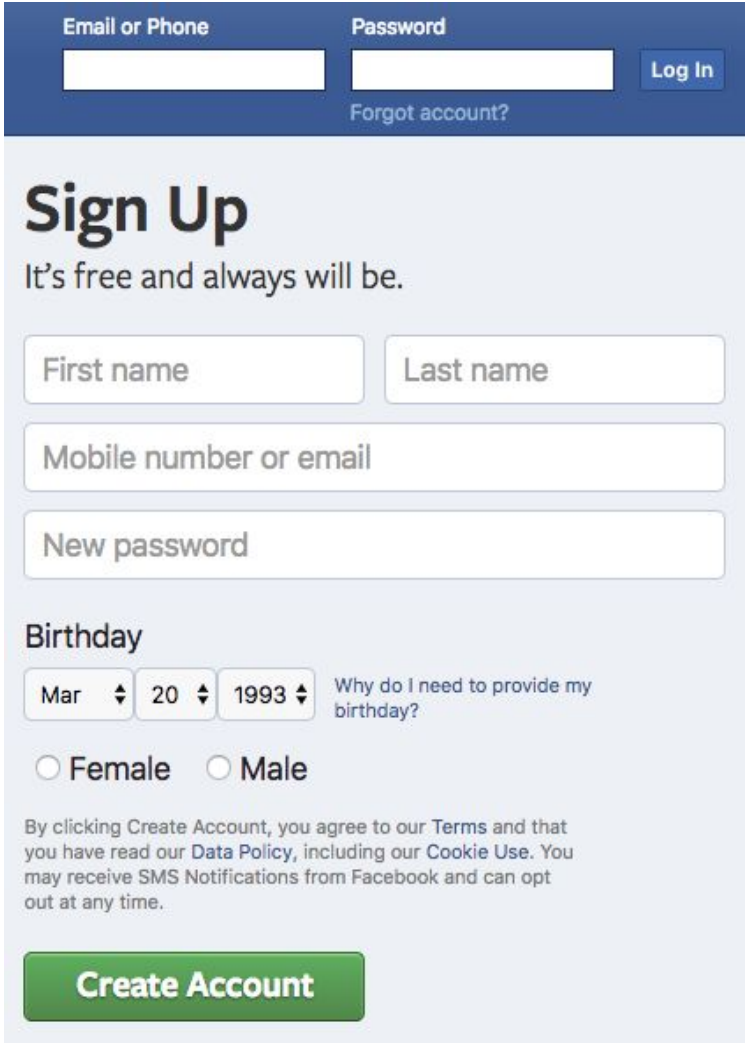
- REST rules dictate that you use GET, POST, PUT, PATCH, DELETE.
- In real world, for most part, only GET and POST are used.
- Sometimes you might see PUT and DELETE.
- Why? Historical design decisions related HTML Forms and Web Browsers.

POST Method

- Used to modify data on a server. Used in conjunction with an HTTP Form.
- Think account login or filling out a survey.
- HTTP POST Requests are formatted almost the same as HTTP GET REquests.
- Parameters are not included in the URI though.
- Stored in the Request Body like GET, but formatted differently.

Facebook Login/Account Creation

- When the “Log In” or “Create Account” button is clicked, the web browser will generate a POST request based on the form data and send it to Facebook for processing.
- Facebook then responds with an HTTP Response.



Note on GET vs. POST methods

- If a server supports it, the client could send data through either method.
- This is strongly NOT encouraged.
- Convention is to stick with GET for getting a resource and POST for updating a resource.
- If you had the following keys-value pairs:
  - Name: Kelvin
  - Hair: Black
  - Eyes: Black
  - Feelings: None

Note on GET vs. POST methods cont.

- For GET request, these could be found in the URI and/or Request Body:
  - Request Body format:

    Name:Kelvin&Hair:Black&Eyes:Black&Feelings:None

- For POST request, these would be found ONLY in the Request Body.
  - Request Body format:

    Name=Kelvin
    Hair=Black
    Eyes=Black
    Feelings=None

HTTP Response Format

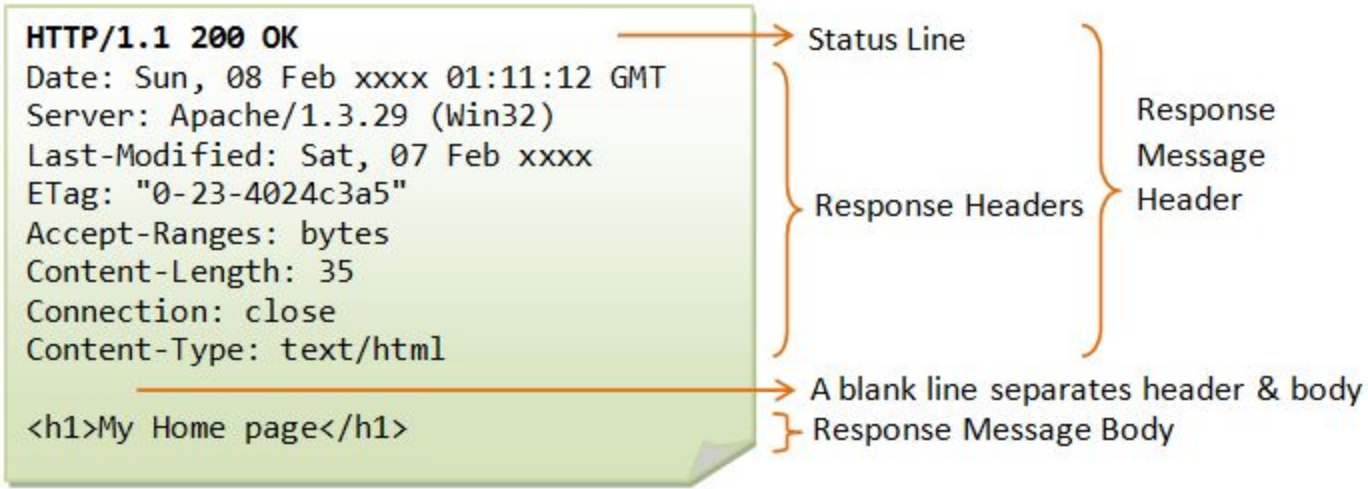


Image Ref: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Status Line

status-line = HTTP-version SP status-code SP reason-phrase CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

HTTP Status Line

status-line = HTTP-version SP status-code SP reason-phrase CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

HTTP Status Line

status-line = HTTP-version SP status-code SP reason-phrase CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

HTTP Status Line

status-line = HTTP-version SP status-code SP reason-phrase CRLF

GET / HTTP/1.1

HTTP/1.1 200 OK, Resource Data...

CRLF?

- CRLF is how the Client knows the response status line has ended.
- What comes next?
  - Response Headers
  - Response Body

HTTP Status Codes

- 1xx – informational message
  - 100 - Continue.
- 2xx – success
  - 200 - OK.
- 3xx – redirect somewhere else
  - 304 - Moved Permanently.
- 4xx – client side error
  - 400 - Bad Request.
  - 403 - Forbidden.
  - 404 - Not Found.
- 5xx – server side error
  - 500 - Internal Server Error.

Example GET Request

GET / HTTP/1.1
Host: www.vt.edu

Example Response

HTTP/1.1 200 OK
Date: Sat, 15 Oct 2016 22:16:51 GMT
Server: Apache
X-RouteInfo: cmsw-prod-01
Cache-Control: max-age=60, public, must-revalidate
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta name="created" content="2016-10-14T07:27:05Z"/>

Example POST Request

POST /fake_login HTTP/1.1
Host: www.vt.edu

username:kaviles
password:iloveece123

Example Response

HTTP/1.1 200 OK
Date: Sat, 15 Oct 2016 22:16:51 GMT
Server: Apache
X-RouteInfo: cmsw-prod-01
Cache-Control: max-age=60, public, must-revalidate
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8

Multipart Requests or Response

- A header that tells the client or server that the request or response will come in multiple parts instead of a single part.
- This is typically for large file upload or downloads.

Caching

- Client can store server response to prevent from making requests again.
- Server response should have information about how caching can be done at the client.
- This is usually done to lighten server load or make client experience load faster.
- Cache control headers
  - **Public** - resource is cacheable by any component
  - **Private** - resource is cacheable by only client and
  - server
  - **No-cache/no-store** - resource is not cacheable
  - **Max-age** - valid up to max-age in seconds
  - **Must-revalidate** - revalidate resource if max-age has passed

Cookies

- HTTP is stateless. But can maintain state by using cookies.
- A cookie is a small piece of data that the server sends back to the client as a result of a request that the client stores
- On subsequent requests to the server, the client automatically includes any cookies that it received from that server
- Example: Language preference for a website

Back to REST

REST Properties

- All content (txt, html, jpg, mp4 etc.) is treated like a resource.
- Must use Universal Resource Identifiers (URIs).
  - Format:
  - Note: Anything in brackets is optional.

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

- Resources can be represented any way. Most commonly JSON and XML.

XML vs. JSON

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40</age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

Image Ref: https://www.sunnyhoi.com/json-vs-xml-format-use-api/

REST Constraints

- Uniform Interface
  - Use HTTP verbs (GET, PUT, POST, DELETE) as action and URI as resource name
- Client-Server Model
  - Decoupled, disconnected client-server system is considered ideal.

REST Constraints

- Stateless
  - Server doesn’t store any state information
  - State is maintained on client’s side
  - Each request is self-descriptive
  - Minimal implementation effort and system overhead

- Cacheable
  - Implicit: server do not specify anything
  - Explicit: server specifies age of cacheable object
  - Negotiated: negotiation can be done between client and server on how long can a resource be cached.

REST Constraints

- Layered System
  - Many layers of hardware, software and middleware is involved
- Code-on demand (optional)
  - Server can extend client by transferring logic or executable to the client as representation
  - Example: Java Applets, JavaScript
- Services that commit to these constraints (except Code-on demand) can be called a fully RESTful web services.

List of things to checkout:

- https://resttesttest.com/
- https://www.getpostman.com/
- Linux curl command
- Python3 requests
- REST/HTTP Requests calls can be made in each of these.
- Chrome console

References

- Roy Fielding:
  - http://roy.gbiv.com/
- Roy Fielding Dissertation:
  - https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- HTTP Methods
  - https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
- HTTP 1.1 Status Codes:
  - https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
- REST Constraints:
  - https://restfulapi.net/rest-architectural-constraints/