
WWW

Web and HTTP


GET /index.html
HTTP/1.1
HTTP/1.1 200 OK
"Welcome to our
Web site!"

Outline

1. Web and HTTP
2. Web API
3. Web Dev Big Picture


WWW

Web and HTTP

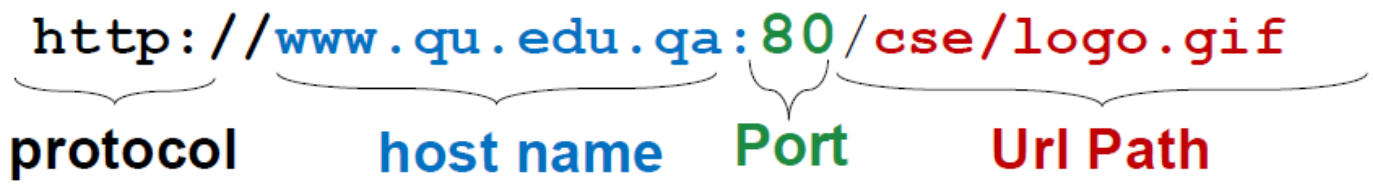

GET /index.html
HTTP/1.1
HTTP/1.1 200 OK
"Welcome to our
Web site!"

What is Web?

- Web = **global distributed system of interlinked hypertext documents accessed over the Internet using the HTTP protocol to serve billions of users worldwide**
  - Consists of set of **resources** located on different servers:

    HTML pages, images, videos and other resources
  - Resources have unique **URL** (Uniform Resource Locator) address
  - Accessed through standard protocols such as HTTP
- The Web has a Client/Server architecture:
  - **Web server** sends resources in response to requests (using HTTP protocol)
  - **Web browser** (client) requests, receives (using HTT protocol) and displays Web resources

Uniform Resource Locator (URL)



- URL is a formatted string, consisting of:

– **Protocol** for communicating with the server (e.g., http, ftp, https, ...)

– **Name of the server or IP** address plus port (e.g. qu.edu.qa:80, localhost:8080)

– **Path of a resource** (e.g. **/directory/index.php**)

– **Parameters** aka **Query String** (optional), e.g.

https://www.google.com/search**?q=qatar%20university**

URL Encoding

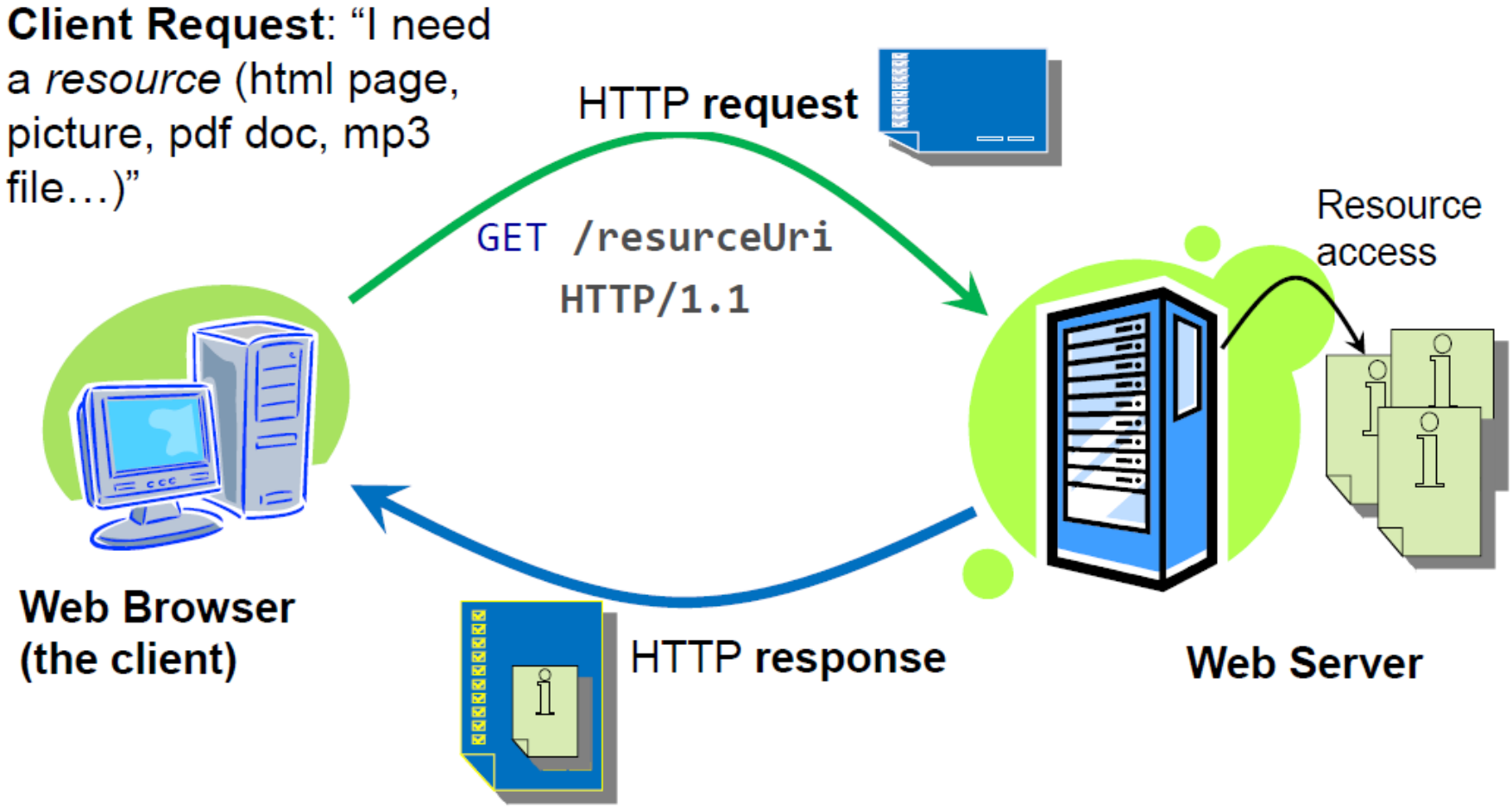
- According RFC 1738, the characters allowed in URL are alphanumeric [0-9a-zA-Z] and the special characters $-_.+!*'()
- Unsafe characters should be encoded, e.g.,

  http://google.com/search?q=qatar%20university

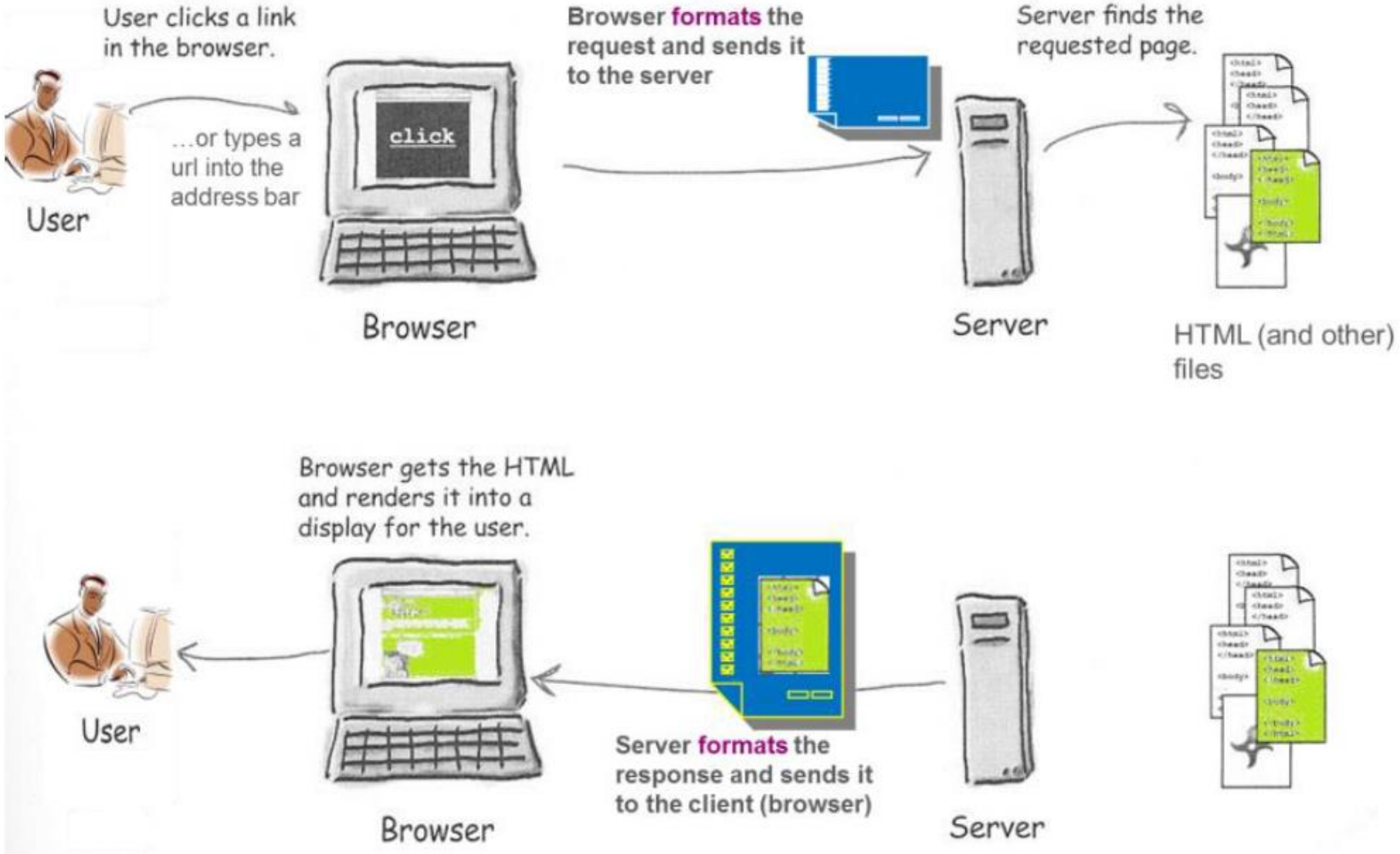
Commonly encoded values:

ASCII Character	URL-encoding
space	%20
!	%21
"	%22
#	%23
$	%24
%	%25
&	%26

Web uses Request/Response interaction model
HTTP is the *message protocol* of the Web

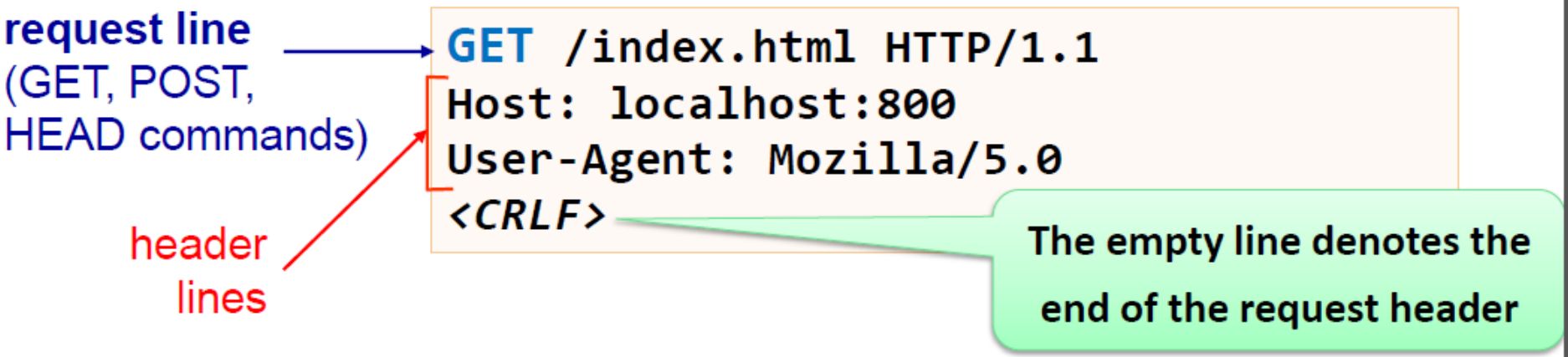


The sequence for retrieving a resource

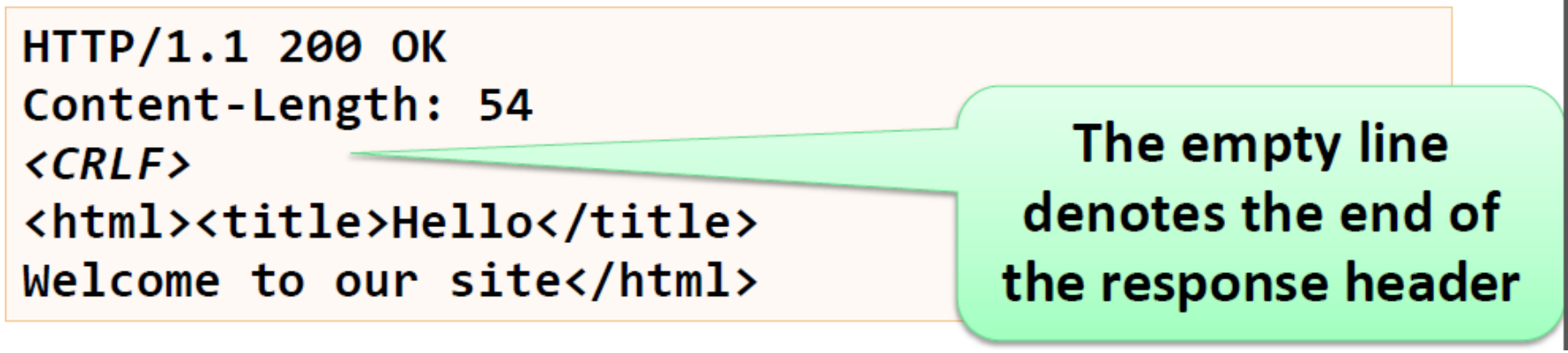


Request and Response Examples

- **HTTP request:**



- **HTTP response:**



HTTP Request Message

- Request message sent by a client consists of

– **Request line** – request method (GET, POST, HEAD, ...), resource URI, and protocol version

– **Request headers** – additional parameters

– **Body** – optional data

  - e.g. posted form data, files, etc.

```
<request method> <URI> <HTTP version>
<headers>
<empty line>
<body>
```

HTTP Request Methods

- **GET**
  - – **Retrieve a resource** (could be static resource such as an image or a dynamically generated resource)
  - – Input is appended to the request URL E.g.,

  http://google.com/?q=Qatar
- **POST**
  - – **Create or Update a resource**
  - – Web pages often include form input. Input is submitted to server in the **message body.** E.g.,


```
POST /calc HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
num1=20&operation=*&num2=10
```

HTTP Response Message

- Response message sent by the server

– **Status line** – protocol version, status code, status phrase

– **Response headers** – provide metadata such as the Content-Type

– **Body** – the contents of the response (i.e., the requested resource)

```
<HTTP version> <status code> <status text>
<headers>
<empty line>
<response body>
```

HTTP Response – Example

status line (protocol status code status text)

Try it out and see HTTP in action using **HttpFox**

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: QU Web Server
Content-Length: 131
<CRLF>
<html>
  <head><title>Calculator</title></head>
  <body>20 * 10 = 200
      <br><br>
      <a href='/calc'>Calculator</a>
  </body>
</html>
```

HTTP response headers

The empty line denotes the end of the response header

Response body. e.g., requested HTML file

Common Internet Media Types

- The **Content-Type** header describes the media type contained in the body of HTTP message
- **Full list @** http://en.wikipedia.org/wiki/MIME_type
- Commonly used media types (**type/**subtype):

Type/Subtype	Description
application/json	JSON data
image/gif	GIF image
image/png	PNG image
video/mp4	MP4 video
text/xml	XML
text/html	HTML
text/plain	Just text

HTTP Response Codes

- Status code appears in 1st line in response message
- HTTP response code classes
  - 2xx: success (e.g., “200 OK”)
  - 3xx: redirection (e.g., "302 Found")

  "302 Found" is used for redirecting the Web browser to another URL
  - 4xx: client error (e.g., “404 Not Found”)
  - 5xx: server error (e.g., “503 Service Unavailable”)

Popular Status Codes

Code	Reason	Description
200	OK	Success!
301	Moved Permanently	Resource moved, don't check here again
302	Moved Temporarily	Resource moved, but check here again
304	Not Modified	Resource hasn't changed since last retrieval
400	Bad Request	Bad syntax?
401	Unauthorized	Client might need to authenticate
403	Forbidden	Refused access
404	Not found	Resource doesn't exist
500	Internal Server Error	Something went wrong during processing
503	Service Unavailable	Server will not service the request

Browser Redirection

- HTTP browser redirection example

– HTTP GET requesting a moved URL:

(Request-Line)	GET /qu HTTP/1.1
Host	localhost:800
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:27.0) Gecko/20100101 Firefox/27.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

– The HTTP response says that the browser should request another URL:

(Status-Line)	HTTP/1.1 301 Moved Permanently
Location	http://qu.edu.qa

Typical server steps to process an HTTP Request

- Parse the HTTP request (i.e., convert a textual representation of the request into an object)
- Generate a response either static one by reading a file or a dynamic response
  - Dynamic response could be either generated programmatically from scratch or it could be generated by filling-up a page template read from a file
- Send the response to the client including:
  - Response headers
  - Response body

2

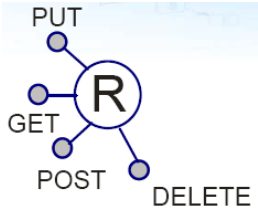
Web API
(aka REST Services)

What is a REST Service?

- Web API = Web accessible Application Programming Interface. Also known as REST Services.
- Web API is a web service that accepts requests and returns **structured data** (JSON in most cases)
  - Programmatically accessible at a particular URL
  - You can think of it as a Web page returning json instead of HTML

- Major goal = **interoperability between heterogeneous systems**

REST Principles



- **Addressable Resources** (nouns): Identified by a URI

(e.g., http://example.com/customers/123)

- **Uniform Interface** (verbs): GET, POST, PUT, and DELETE

  -Use verbs to exchange application state and representation

  -Embracing HTTP as an Application Protocol

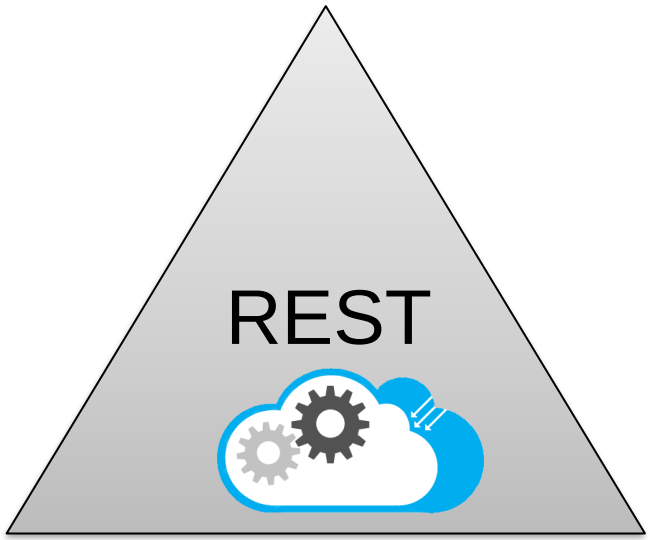
- **Representation-oriented**

  -**Representation of the resource state** transferred between client and server in a variety of data formats: XML, JSON, (X)HTML, RSS..

- **Hyperlinks** define relationships between resources and valid state transitions of the service interaction

REST Services Main Concepts

Nouns (Resources)
e.g., http://example.com/employees/12345



Verbs
e.g., GET, POST

Representations
e.g., XML, JSON

Resources

- The key abstraction in REST is a **resource**
- A resource is a conceptual mapping to a set of entities
  - Any **information that can be named can be a resource**: a document or image, a temporal service (e.g. "today's weather in Doha"), a collection of books and their authors, and so on
- Represented with a global identifier (URI in HTTP)
  - http://www.boeing.com/aircraft/747

Naming Resources

- REST uses URI to identify resources

> Dedicated **api** path is recommended for better organization

  - http://localhost/**api**/books/
  - http://localhost/api/books/ISBN-0011
  - http://localhost/api/books/ISBN-0011/authors

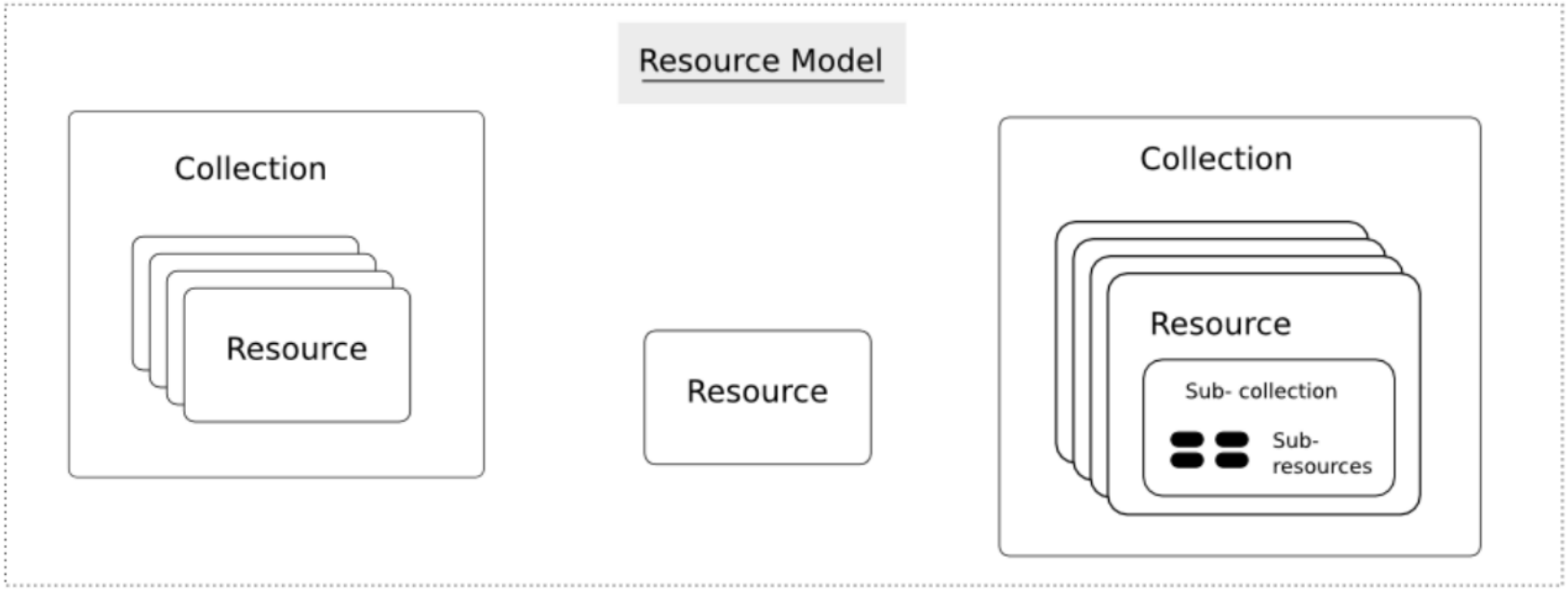
  - http://localhost/api/classes
  - http://localhost/api/classes/cmps356
  - http://localhost/api/classes/cs356/students

- As you traverse the **path** from more generic to more specific, you are navigating the data

Example CRUD (Create, Read, Update and Delete) API that manages books

- Create a new book
  - **POST** /books
- Retrieve all books
  - **GET** /books
- Retrieve a particular book
  - **GET** /books/:id
- Replace a book
  - **PUT** /books/:id
- Update a book
  - **PATCH** /books/:id
- Delete a book
  - **DELETE** /books/:id

A Collection with Resources



A Collection with Resources

A Singleton Resource

Sub-collections and Sub-resources

Representations

Two main formats:

- **JSON**

```
{
    code: 'cmp123',
    name: 'Web Development'
}
```

- **XML**

```
<course>
    <code>cmp123</code>
    <name>Web Development</name>
</course>
```

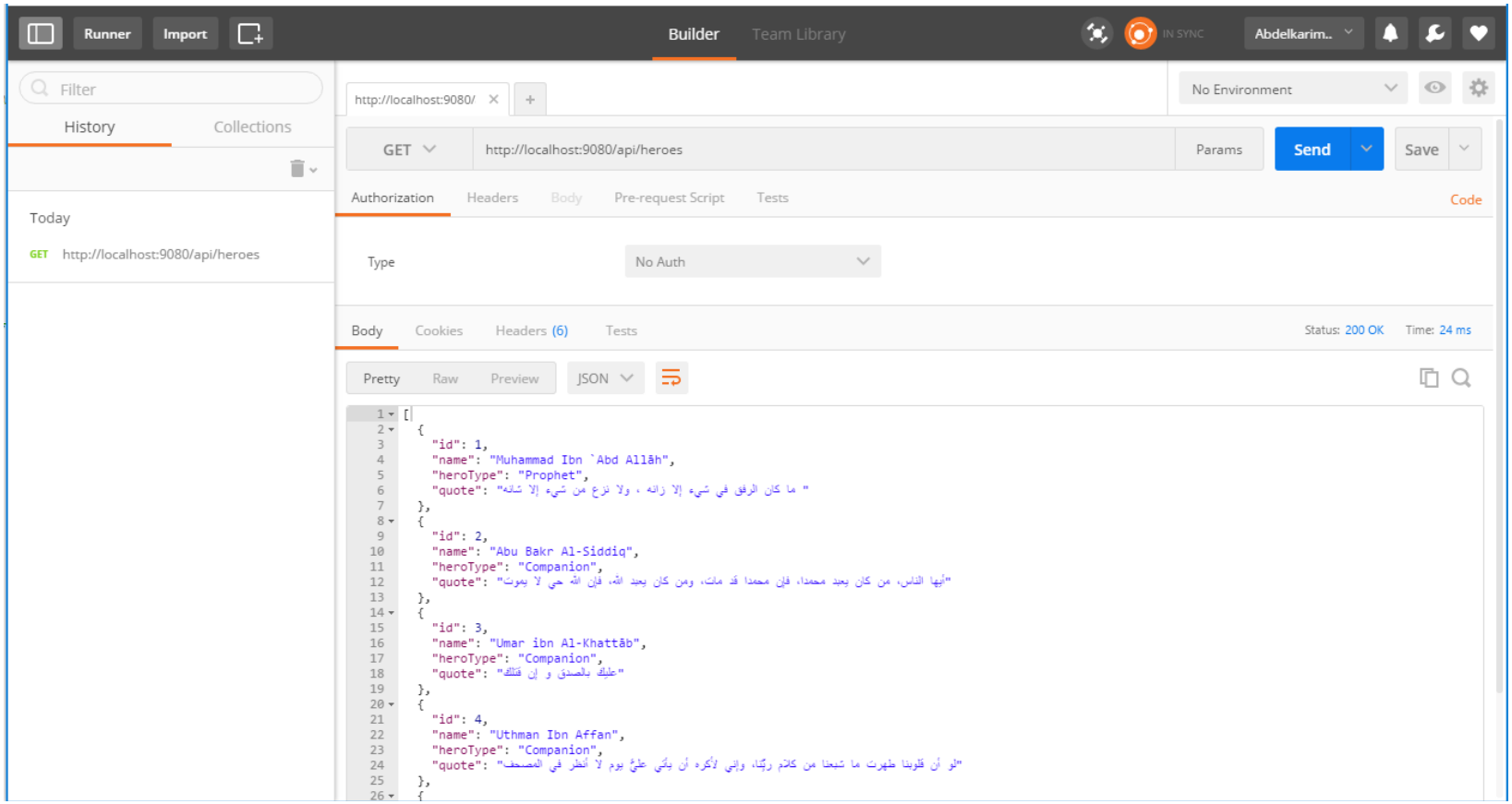
HTTP Verbs

- Represent the actions to be performed on resources
- Retrieve a representation of a resource: **GET**
- Create a new resource:
  - Use **POST** when the server decides the new resource URI
    - Post is not repeatable
  - Use **PUT** when the client decides the new resource URI
    - Put is repeatable
- **PUT** is typically used for update
- Delete an existing resource: **DELETE**
- Get metadata about an existing resource: **HEAD**
- See which of the verbs the resource understands: **OPTIONS**

Testing REST Services

- Using Postman

https://www.getpostman.com/



Web Dev Big Picture

Web Client

Request

Response

Web Server

Frontend development

HTML for page content & structure

CSS for styling

JavaScript for interaction

Backend development

Web API

Web Pages

Data Management