The web

The World Wide Web is an information space made up of two chief components: resources (mostly stored on servers), and the entities that request those resources (usually called clients). When you visit a website, your browser acts as a client. It requests a resource - e.g. the google homepage - from a server via a transmission protocol ( e.g. HTTP). The server returns the requested resource. This is called the client-server relationship.

A close up of a logo

Description automatically generated

To make a server, a socket is created by the server computer on its network listening on a specific port. Open sockets transfer data of a certain transmission protocol (HTTP / FTP / Telnet) from a specific port to the program running on the computer, this allows the computer to read the request. Since servers also have the reply, sockets can be used to send data from a computer to another network address&port.

It is possible to create a socket on Ruby requiring ‘socket’ then creating a new server using ‘ server = TCPServer.new(<port-num>)’ and using that instance to create a socket by ‘socket = server.accept’. To listen for text data use ‘socket.gets’ (each line of a request will be received as a new line, so to get all lines of a HTTP request, use a loop until lines go blank) and to send text data use ‘socket.print’ or puts.

URL

URL - Uniform Resource Locator

Used to identify a resource of the web, normally web address which can be convereted by DNS. It has four sections: *protocol://hostname:port/path-and-file-name*

1. Protocol - Application layer (HTTP / FTP / telnet)
2. Hostname - DNS name or IP of server ([www.google.com](http://www.google.com) / 174.191.183.123)
3. Port - Port to connect on (TCP or UDP)
4. Path and filename - Name and location of resource on server (public/doc.txt)

URLs cannot contain special charaters such as spaces or ~ so they will get escaped using % and ASCII hex coded and converted if required.

URI

URI - Uniform Resource Identifier

More general than a URL and can locate a fragment within a resource. The URI syntax is:

*http://host:port/path?request-parameters#nameAnchor*

* Request parameters are seperated from the URL by a ‘?’
* name:value pairs are linked with ‘&’
* a name anchor identifes a fragment within a HTML document using *#nameAnchor* a the end of the name:value pairs
* Session management can be performed wth ‘;sessionID=xxxxx’

HTTP

The Hyper Text Transfer Protocol is the main transfer protocol used by clients to talk to webpages on the internet. It is called a transfer protocol since there is a defined structure for requests and responses. It is a stateless transfer protocol as it assumes very little about a system and does not keep state between different message exhanges. HTTP is normally transmitted between computers using the TCP/IP protocol over port 80.

Communication occures via request/response pair. Within requests the included data send is called the parameter which come in key-value pairs. One way of passing a parameter to a server is passing it in the **query string**, this is where the parameter is appended to the end of the URL after a ‘?’, mutlipul parameters can be chained using the & symbol. e.g. here the key-value pairs are highlighted.

<http://www.example.com/home?name=Bob&age=21>

Uniform Resource Locators (URLs) reveal the identity of the host wanting to be communicated to but the specific actions are specified via HTTP request verbs. Some standard verbs are:

* GET - fetch exisiting resource (use for safe actions), some data passed in URL query string
* POST - create exisiting resource from data provided (use for unsafe actions), arbitrary amount of any data passed in body of request message
* PUT - update exisiting resource with data provided
* DELETE - delete exisiting resource
* HEAD - similar to get without message body, so gets server headers
* TRACE - used to retrive hops that a request takes to round from from the server, proxy adds IP or DNS name
* OPTIONS - retrives server capabilities

When the client initiates request to the server, in return the server responds with status codes and a payload . The status code tells the client how to interpret the response.

HTTP Generic Structure:

A screenshot of a cell phone

Description automatically generated

HTTP Request Message Parts

* Request Line - Type of request, URI (Resource requested), HTTP Version (1.0 or 1.1)
* Headers - Optional list of name:value pairs such as host, connection type, language
* Body - Optional body, contains key:value pairs in post request or any other included data

A screenshot of a cell phone

Description automatically generated

HTTP Response Message Parts:

* Status Line - HTTP Version, Status Code, Reason
* Headers - name:value pairs of headers such as content length, last modified etc
* Body - Data packet normally HTML code

A screenshot of a cell phone

Description automatically generated

<https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177>

URL parts - <https://www.tutorialspoint.com/html/understanding_url_tutorial.htm>

<https://doepud.co.uk/blog.php/anatomy-of-a-url>

RESTful Routes

The web is a network of resources, data stored somewhere e.g. bookmark, image, etc. A website is a collection of resources, aptley named Collections. Representations tie URL’s to actions a users might want to take with a resource, e.g using a route /bookmarks/1 to return the bookmark with ID 1.

REST = Representational State Tranfer

RESTful routes is a set of conventions which define a consistent way of creating these addresses to the resources. If the rules aren’t followed the routes will still work, it’ll just be confusing to follow for other users.

RESTful routing shows the app as a state machine, in which there are a finite number of states which can be determined by user interactions. The methods:

* GET - request data and produce no side effects
* POST - create new resource
* DELETE - delete resource
* PUT - update resource

Will allow the user to change the state of the machine (app) e.g. post /bookmarks will increase the state of bookmarks from 35 to 36.

Rules:

* Terminologys - Resource/Collections/URL
* API Endpoint - Each route should name a resource (ie always pural)
* HTTP Methods - Each route should have the HTTP methods to modify that resource in the specified way, e.g. post should make a new resource of its name. e.g.
  + POST /bookmarks should create a new bookmark
  + GET /bookmarks should return a list of bookmarks
  + GET /bookmarks/14 should return the bookmark with ID 14
  + DELETE /companies/3/employees/45 should delete empolyee 45 from company 3
* HTTP Response codes - Error codes conventions e.g. 2xx => success category
* Field name convetion - consistent casing convetions across application
* Searching, sorting, filtering, pagination - These actions simply query one dataset, no new API to handle the actions:
  + Sorting - if client wants sorted companies, GET must accept params which allow for the return of sorted companies in the response, e.g. GET /companies?sort=rank
  + Filtering - Filtering is done by passing params via GET to page which needs filtering
  + Searching - Searching is done by passing params via GET to page which needs searching
  + Pagination - Splitting data into smaller sets i.e. page 2 onwards, should be done by passing page num params to GET e.g. GET /companies?page=23
* Versioning - Version details in the route so that application can still access old APIs if there is an update.