COMP2406 – Introduction to Web Applications and HTTP:

Submitted by: Stefan Majlovic

Basic Desktop App Architecture involves a PC with Input and Output Storage registering input from hardware and displaying this information as an output on consoles.

* The process is user-friendly and streamlined – this shows the core of encapsulation and abstraction, as the person using the computer is only provided with the basic core for use.
* You do not require special tech knowledge to take advantage of this – the result highlights good program practices and keeps processes simple.

Web App Programming uses more complicated components, however:

-Client-server decoupling is simple to access, but harder to comprehend – more detailed understanding of tech processes is required.

-Numerous users simultaneously using the web contrasts it from desktop app use – it makes use of synchronous and asynchronous processes to handle this multitude of data.

-Dominated by event-driven programming makes use of the Model-view-controller design pattern. Web design is an advanced form of GUI creation and expands on this with live manipulation of information instead of controlled GUI programs.

Web App Architecture Process:

The Server’s address that provides information to the clients using the internet is stored remotely in databases. Each aspect interacts with the internet in unique ways:

-Clients access the http address, while the server sends responses, which are directly sent towards clients.

-Clients and servers have unique I.P and DNS addresses to ensure no mistakes are made when sending information (risk of improper data transmission could mean the wrong user getting undesired data)

HTTP handles all the client and server communication and data transferring. The role of HTTP as the middleman for data is valuable, as it lessens the burden of data storage, time-allocation, and transmissions for servers – each component is free to focus on their specialization, making **memory allocation more efficient**. No client could reasonably handle the entirety of data present on a server.

-This frees up room for clients to specify what they desire, and servers can act accordingly through HTTP’s information gathering and transferring.

-Because the internet is so gigantic, the web needs to be incredibly effective with how data is sent, and what is sent to begin with.

-Since so many people can be using a webpage at once, small changes in load time (and thus efficiency) and far more pertinent than in desktop applications – it impacts more people.

-Good programming practices are a must for large-scale web projects and servers.

Client-Side technologies: HTML5, CSS, JavaScript

Intermediary: HTTP

Server-Side: Node.js, NPM, JavaScript, MongoDB

-Each has a particular role, and as such, replicates the MVC paradigm in how information is requested, processed, and later displayed through actions from these technologies.

Mozilla offers an excellent guide on understanding these technologies, and even provides practice for them: <https://developer.mozilla.org/en-US/docs/Learn>

HTTP Basics:

-Stands for Hyper Text Transfer Protocol – responsible for distributed and collaborative hypermedia information systems.

-Hypermedia involves critical web functions such as providing links from one page to another page on the World Wide Web (see the link above as an example).

-Uses a request/response model (servers send responses to user resource requests)

* Data elements such as **headers** provide specific details on requests – the outer layer of visibility.
* Requests may also contain **bodies**, which are the main content of the request (the whole inner parts of the body)
* These act as levels of *hierarchical classification*, and helps optimize search-results (e.g. searching something on a Web Browser – you provide specific information, engine uses this to find what you are looking for)

HTTP is stateless – does not retain information after clients provide their inputs. This means that all useful information must be provided in the initial search, but this is beneficial for information purposes, since it ensures **any server can handle any request from any client**.

-This independence shows the value of specialization – in the event of failure, a server is not sabotaged in any capacity should a client fail to connect, or once a request is handled. Processes are optimized and scalable.

-HTTP protocol uses plain text for requests/responses – though the content can be in other formats. Meant for human readability – it is valuable for people to understand the information being uploaded and stored.

-Of course, this benefit can be a weakness, in the context of malicious users. A secondary layer of protection is necessary to lessen the risk of abuse: DNS.

DNS is like a unique phone-number: it maps domain names to IP addresses – it makes otherwise illegible data readable for users and eliminates the need to remember scores of numbers. Instead of Carleton.ca, you would need to remember 134.117.6.162!

-To understand DNS better, Cloudflare offers a useful guide on the subject: <https://www.cloudflare.com/learning/dns/what-is-dns/>

Uniform Resource Locators (URLs)

-Represent location and name of resource on the web – these request specific resources.

Verbal Example: “I would like Culearn, located in Ottawa, Ontario, and owned by Carleton University to show me the section with my Grades.”

-URLS can be broken down into many parts – again, it is a form of hierarchy: https:// (website URL. Extension)/…other potential webpages

-It can be best understood as like operating a console in your Desktop (cd Documents/Carleton/…)

In order: it accesses the protocol, hostname, port, pathway, querystring, and then fragment.

-Protocols determine how the request is transmitted (who is responsible for this?)

-Hostnames specify the server to be accessed (generally a full domain name, and a top level domain (the extension, .com, .ca, .net, etc.)

Google.com -> google is the domain name, with .com indicating its primary purpose (commercial/general purposes)

Port allows information to be directed to a specific listener within the server – like handing a ticket to a specific person, while awaiting further instructions.

Path dictates what specifically is desired: identifies pages or resources within the app.

Querystrings are optional key/value pairs. Like assigning values to specified keys. A parameter for a function, in a sense. These can be passed when using **request methods (see below)**.

Fragments are not passed to the server – more like markers within a page to return to by the browser.

Considerations:

-URLS are addresses – they are the space within the World Wide Web that people can seek to access and acquire in computer networks.

-URLS specify their location – a URL accepts that it can be publicly accessed, and often times can be accessed freely if not privately restricted or hidden.

-Without URL’s, it would be like trying to find a specific house in a country with no map or guide.

HTTP Requests: these have the following format

-A request line (describes what needs to be implemented)

-Zero or more headers (specifics to locate or traverse)

-A blank line (indicates the end of the header fields)

-An optional message body (some additional information to search)

See <https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html> for more information.

Request Methods:

GET – retrieves a document (cacheable, stays in browser history, bookmarkable, can only **request data** and primarily for public or non-sensitive information)

-If it is valuable, a malicious entity could take advantage of its easily accessible and savable nature to conduct theft or attacks.

-Consider the nature of public and private abstraction in Java to better understand why we want to hide important data.

POST – sends data to the server to create/update it (never cached, don’t remain in browser history, cannot bookmark requests)

HEAD – like GET, but primarily retrieves just the headers

PUT – stores new resources or replaces one

DELETE – removes a resource

W3Schools has a useful guide on how to use these in further detail (<https://www.w3schools.com/tags/ref_httpmethods.asp>).

-Request Methods are ways of interaction with URL’s. A way to have specific decisions made for URL. A URL would not be very beneficial, if one could not modify it for some purpose.

Static vs Dynamic: if it were static, changes would only occur if directly changed in the code! Dynamic URL’s are more manipulatable – but less friendly to search engines therefore (the search engine would need to account for constant changes to it).

-So, request methods are helpful to indicate to the server what should specifically be processed and sent back.

Common Request Headers:

Accept – specify what content type(s) to accept

Accept-Encoding – specify what compression

Authorization – used for HTTP authentication

Content-Length – the length of the body

Content-Type – the content type within the body

User-Agent – requesting user agent (e.g., browser)

-It is valuable to have many options for request headers, since we want to account for potentially many types of requests that could be made. Not all requests are equivalent in importance, so ways to specify what is desired are beneficial for optimal retrieval.

HTTP Response:

-These contain headers that specify properties of responses, just like requests. They also contain a body with the response data. This provides pertinent information for the client and helps identify what the server knows (and will share) given the inserted requests properties.

-A response would contain the HTTP Version (HTTP/1.1), a status code (404, 504), status text (corresponding to the status code)

-Response Headers to provide specific information (such as date of response, server, content-length, and such)

-A blank line, to indicate the end (equivalent to how requests have one) and finally, a response body.

Response Status Codes:

1xx – Informational

2xx – Great success!

3xx – Redirection

4xx – Client error

5xx – Server error

-These are computer-readable – a human would not necessarily make sense of this without prior knowledge.

Common response headers:

Content-Type – type of content in the response body

Content-Length – length of the response body

Cache-Control – the type of caching allowed

Content-Encoding – the compression used

Expires – the date at which cached data expires

Last-Modified – the last time the data was modified

-Response headers use these parameters to provide specification to their responses – these are a general outline of what to expect when a response is made.

-Specification is helpful not only for the computer, to know what kind of response header is sent, but also for a person that is looking for specific details relating to it. These are mainly useful for verification, and to ensure the response is working as intended.

HTTP Interaction Steps:

1. Browser/Clients send an HTTP request for a particular html page.
2. The server, noting a request to access it, makes its decision and sends an HTTP reply containing the html or denies such.
3. Browser scans the HTML for resources (safety precaution, provides additional knowledge of what should be presented/loaded, etc.)
4. Server sends HTTP replies if specific elements are requested by the browser (such as wishing to upload a JPG file)
5. Browsers use these resources to ‘render’ the original requested page.

-A systematic process, interactions between servers and clients show the importance of proper memory allocation – a poorly designed server could take a long time to validate requests for clients.

-As such, avoiding transmissions of information is desirable when possible. This is the basis for caching – storing information on the client, saving time and space for further interactions.

Additional Material:

Cookies are information stored in a client, so that a server will not need to transmit it as much later. You want each client to have their own unique basket filled with information, rather than one giant basket shared among them.

-In essence, a cookie helps identify a specific client.

-Note, however, that **cookies are not exactly caching – more so an alternative to avoid it.**

Return to Caching:

-CSS stylesheets, JavaScript files, and many images do not change often – inefficient to send after being sent once if they have not been changed. In other words: duplicates are avoided by simply sending once. Multiple copies of the same image would be cached automatically (no point to send unique versions of multiple copies of the same image, if nothing is different)

-Servers can control caching – by using response headers such as Cache-Control. The default is public if nothing is specified.

Types of Caching:

no-cache: the content cannot be re-used without verifying with server

private: the content can be cached on the local machine

public: the content can be cached in public caches (e.g., proxy servers, CDNs)

no-store: the content cannot be cached (e.g., for private data you don’t want to stick around)

Considerations: Like Java and abstraction – we want to control what the user can access and modify (public, private specifications). A server does not want a client accessing highly critical information if it is necessary for the webpage’s rendering.

-Content age can be specified – how long do you want information cached?

e.g. private, max-age = 3600 (cached for 1 hour on a local machine)

-Each digit represents 1 second.

Finally, there are headers to specify the data the content was last modifies, as well as when it expires.

This can be used to make conditional requests.

-This means that we can access states – we can access when content has been changed (or if it has been since x date). This only informs us with headers – the data is not re-sent.

-This process is efficient and concise – only providing the critical information necessary for understanding the outcome.

Resources:

<https://developer.mozilla.org/en-US/docs/Learn>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>

<https://www.cloudflare.com/learning/dns/what-is-dns/>

<https://www.techopedia.com/definition/1352/uniform-resource-locator-url>

<https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html>

<https://medium.com/techblogout/whats-the-difference-between-cache-and-cookies-53e7f4f094bb>