Brian Chen

CSCC09 – Programming on the Web
University of Toronto Scarborough, Winter 2019

Introduction

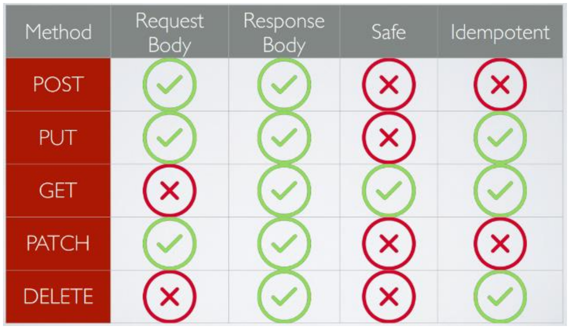
Architecture – Client sided (web browser) and Server sided (web server)
Virtuous Cycle – faster better technology ⇔ new usages
AJAX (interactivity) **HTML5 (**multimedia), increased speed of JS, rendering, homogeneous implementations, frameworks
Data Storage and data processing are moving from desktop to the Cloud
Rich Content + Cloud Computing = new way to think about software systems (web tech is heart of this change)

XHTML + CSS

HTML – Content   CSS – Presentation   JavaScript – Processing
HTML 4 – Separation between content and presentation
HTML 5 – Multimedia (2008)
<Markup> are the tags that start and end with angle brackets **</Markup>**
Content is basically everything else
Element is the start and end tag with content in between   **<p>** hello **</p>**
Attributes are name/value pairs specified in a start tag   **<img src="leader.png"/>**
Comments are tags that will be ignored at rendering   <!—hello! -->
XHTML is redesigned HTML so that its stricter (requires certain tags: html, body, head, title, etc.) and must match bracket
  Deprecated CSS styling in HTML
CSS can be:
  Inline   **<p style="background: blue">**
  Embedded   **<style TYPE="text/css> p{background: blue;} </style>**
  Separate file   **<link rel="stylesheet" type="text/css" href="style.css"/>**
Classes are sets of HTML elements for which we want to define common properties   class="button"   .button{}
IDs are attributes of HTML elements for which we want to identify uniquely   id="sale"   #sale{}

The HTTP protocol

Standard TCP protocol on port 80
URL/URI identifies what resource is being accessed
Request method w/a specified command
http://www.utsc.utoronto.ca/~registrar/index.php?course2=CSCC09H3
^ protocol   ^server   ^path   ^query ^resource ^parameters
POST – add an unidentified resource
PUT – add an identified resource
GET – get a resource
PATCH – update a resource
DELETE – delete a resource   also… HEAD, TRACE, CONNECT, OPTIONS
A HTTP Request has:
  Method
  Query String
  Headers (key/value pairs)
  Body (data, optional)
Curl -v (verbose) –request (request_mthod) –data (request_body) –header (header) **URL**
HTTP Responses –
  Status code
  Headers
  Body (optional)
1xx – info   2xx – success   3xx – redirection   4xx – client error   5xx – server errors
Safe – may not have side effects
Idempotent – same result when called multiple times



Method	Request Body	Response Body	Safe	Idempotent
POST	✓	✓	✗	✗
PUT	✓	✓	✗	✓
GET	✗	✓	✓	✓
PATCH	✓	✓	✗	✗
DELETE	✗	✓	✗	✓

JavaScript on the Server

Node.js runs on Chrome V8 Javascript engine
Non-blocking IO, no restrictions
  Ran by **node example.js**
Routing can be done but is tedious, so we often use Express.js
Sending data structure between frontend/backend: **URI/JSON Encoding**

JavaScript Object Notation (JSON)

JSON Standard – lightweight open format to interchange data, since 2009 most browsers support JSON natively
JSON is either:

- Indexed array (array) [1, 2, 3, 4, 5]
- Associative array (object) [{"name": "Thierry"}, {"name": "Jeff"}]

Serialization: JavaScript → JSON var myJSONText = JSON.stringify(myObject);
Deserialization: JavaScript ← JSON var myObject = JSON.parse(myJSONText);

AJAX – Asynchronous JavaScript and XML

Fetching data without refreshing a page
It is not a **language** but a simple **javascript command**
Good because its low latency and rich interactions, but puts center of gravity on client side instead
var xhr = new XMLHttpRequest();
xhr.open(method, url, true); ← the true is the asynchronous part
Some Concurrency Issues!

REST (Web API)

The server side is more or less a **storage system**
REST is **Representational State Transfer**
Designing a remote APi for a storage system using HTTP

Function Names – method and URL
Function Arguments – URL and request body
Returned Value – status code and response body

Has one-to-one, one-to-many and many-to-many relationships
CRUD – Create, **R**ead, **U**pdate, **D**elete

Storing Data

Persistency, Concurrency, Query, Scalability
Relational (SQL) databases – tables and tuples, uses SQL, inadequate for big data, ACID transactions, PostGres, MySQL
NoSQL databases – key/value pairs, API style, lack of consistency, adequate for big data, MongoDB, redis, neDB, etc.
Object Relational Mapping – mapping between OOP and database structure, Mongoose/Sequelize
Retrieve Selected Items only (instead of whole collection)
Define primary keys (instead of autogenerating)
Split data into different collections
Create join collections whenever appropriate

Handling Files

Can't get file unless user specifies
Can send a file through form action, or AJAX requests
Server get file metadata (filename, file type [mimetype], size, etc.) and file content (compressed binary or string)
MIMETypes – Multipurpose Internet Mail Extensions, content-type (text/html, text/css, image/jpeg, application/pdf, etc.)
Don't send base64 files using JSON, encode it instead and compress it using multipart/form-data
Do not store uploaded files with static content, or serve them statically
Do store the mimetype and send it back with the files

Cookies and Sessions

Cookies are key/value pairs of data sent back and forth between the browser and the server in HTTP request and response
Text data (up to 4KB), may/may not expire, can be manipulated from client Document.cookie **and** server (express cookie)
Sessions are session id (aka token) between the browser and the web application, should be **unique** and **unforgeable**

Web Authentication

Local auth using login/password (store using salted hash)
Token-based auth
Third-party auth
Send passwords in header, body, never in the URL
Store passwords as salted hash only

Token-based auth user's secret and some request arguments, password never transit back/forth, digest can be sent clear

---

Single Sign On (SSO) – using Pubcookie, OpenId, SAML, **OAuth**, signs in using third party page (facebook, etc.) gives a token. Verifies token w/third party, starts a session.

Web Security

Top Vulnerabilities – Information leakage, Cross site scripting, content spoofing, insufficient transport layer protection, cross site request forgery
Insufficient Transport Layer Protection – Use HTTPS. Attackers can eavesdrop in messages (secret exchange of info), tamper with messages (reliability)
HTTPS – HTTP + TLS (transport layer security), provides **confidentiality** (end to end secure channel) and **integrity** (auth handshake)
Self-signed certificates are not trusted by your browser. Browser trusts **Certificate Authorities** by default – can be found.
HTTPS must be used during an entire session, not only selectively
Secure cookie flag makes it so that the cookie will only be sent over HTTPS, helpful against mixed-content shenanigans
HttpOnly cookie flag makes it so that the cookie is not readable/writable from the frontend
Samesite cookie flag makes it so cookie will not be sent over cross-site requests

---

The backend is the only trusted domain – sensitive operations must be done only on the backend
SQL Injections – attackers can inject SQL/NoSQL code, retrieve/add/modify/delete info & bypass authentication
Insertion into HTML → **The data inserted into the DOM must be validated**

Cross-site scripting – XSS is JavaScript Code Injection. Problem is attacker can inject arbitrary js code that can be executed by the browser. → **The data inserted into the DOM must be validated**

> **Reflected XSS –** malicious data sent to backend are sent to frontend to be inserted into DOM
> **Stored XSS –** malicious data are stored in database and later on sent back to frontend and inserted into the DOM
> **DOM-based attack** malicious data are manipulated in frontend and inserted into the DOM

Cross-site request forgery
Same origin policy → resources must come from the same domain (protocol, host, port) (AJAX requests, form actions)
Digression – relaxing the same-origin policy
We can **protect legitimate requests using a CSRF token**

Deploying a Web Application / Building Fast WebApps

Running the application on a server that is connected to internet and always powered. Need domain + HTTPS
OS / Webserver / Database
LAMP – linux, apache, mysql, php/perl/python
MEAN – mongodb, express, angular/react, node
Web Host – a home for your website (OS/server, etc.)
Domain name – URL for your website
Valid certificate – a signed certificate for HTTPS
Development servers do not scale, need to set up a production server
Considerations: Storage (how much space), bandwith (traffic), and money (how much spent daily)
Physical servers, virtual private servers, shared web hosts
You need to buy a domain name from a Domain Name Registrar (namecheap, godaddy, etc.)

Optimizing backend code with **web caching** and **scaling over multiple servers**
Static content -> http proxy cache (intermediary that serves static files before webapp, can cache static stuff)
Dynamic content -> memory cache, controlled by the program (stuff like Memcached)
Cache Stampede – multiple concurrent requests of same request because cache was cleared. Needs cache warming
Load balancer can distribute the load over multiple servers
CDN – content distribution network, distribute the load based on whos active in the world

---

Backend templates – getting static/dynamic/api all in one go (good for more reads than writes, e.g. Reddit). Code reuse and faster loading time (avoids unnecessary AJAX). They are built on the server and retrived through 1 HTTP request, and can be cached on the server as well.
Frontend packing – packs files together, removes whitespace, shortens variable names (webpack.js)
HTTP2 – send multiple HTTP responses for a given request,
Short/long polling – short polling requests update every few seconds, replies regardless of if theres an update. **Long polling** only replies when there is an update.

Advanced JavaScript

All callbacks are executed separately, but in reality its single threaded.
Promises just resolve callback hell, we have a | structure
Async/wait is built on top of promises. var data = await readFile(filepath); ← waits til it finishes
JavaScript event loop – when the stack is cleared it checks the event loop (by webapi or other) for other events to do
Webworker – creates threads in Javascript, duplicating JS event loop, can run in parallel, can do XMLHTTP, indexedDB, location, etc. But cannot access window or document

Internationalization (i18n)

Internationalization (I18N) is the process of designing software so it can be adapted to various languages without changes. Language agnostic software.
Localization (L10n) is the process of adapting internationalized software by adding locale-specific components and translating text. Adapting an application for a specific location
Other considerations:

- Number format
- Date/time
- Punctuation
- Sort orders
- Units/conversion
- Currency
- Paper size/layout

Request headers have a field called **accept-language**
Alternative options: storing language in the URL (en.canada.ca), in the user profile, in the cookie, etc.

WebServices

Mostly dead by Google's switch to JSON APIs. We could do web scraping, json, programming APIs, or SOAP messages.
Web Services are implementation of a **remote procedural call (RPC)** over HTTP/etc. This remote procedure is a web service. Mostly used between web services (B2B). **This is a Service Oriented Architecture (SOA).**
Boxes and standards were evolving too fast, that's why web services eventually failed.
SOAP – simple object protocol, provides ways to exchange messages
WSDL – web service definition language, provides a way to describe web services
UDDI – universal definition language, provides a way to advertise web services
Very flexible, very complex, standards evolve faster than frameworks. Ad-hoc principles (REST/JSON) are used.

Advertising on the Web

Click banners – or pay per view, ad revenue for each click on the relevant ad
Sponsored links – buying keyword associations
Web Scraping – a website that will extract collect and aggregate data from other websites. The goal is to attract visitors to your site and fool them to click onto ads.
Click Fraud – bot that automatically clicks on ads displayed on the website, or web.

Log file analysis – server side analyzing the web server logs
Page tagging analysis – JS code analyzing user interactions (google analytics)

Web Tracking – cookies with unique IDs to identify same user visiting different websites
Privacy mode – disable browser data storage – web cache, http cookies, html5 local storage, etc.
Do Not Track – a request to the website in the HTTP header field, web doesn't have to honor that request