Web Programming (CS-A)
CS 406

Lecture 2

Shamsa Abid

Agenda

- HTTP Request Message
- HTTP Response Message
- Web Clients
- Web Servers
- Web Technologies
- Web Application Architecture

HTTP Message

- Message is either:
  - **Request:** from client to server
  - **Response:** from server to client
- Each Message comprises of
  - **Start Line:** defines the message type and other identifying information
  - **Message Header:** contains meta data
  - **Message Body:** contains the actual body of message – generally the entity associated with the message. It may or may not be empty depending upon the Message Type

Entity

- An entity is the content that is intended to be transferred between the client and the server
- Entity may be further divided into:
  - **Header:** That contains meta-information about the body that helps in proper interpretation of body
  - **Body:** The actual content to be delivered
- Examples of Entity:
  - HTML Document
  - XML Document, etc

Example Request/Response Message Cycle

- GET http://www.google.com HTTP/1.1
  Host: www.google.com

Request

- HTTP/1.1 200
  Content-type: text/html
  Content-length: *xx*

  <html>
  <head> … </head>
  <body> Google </body>
  </html>

Response

HTTP Request Message

- Structure
  - Request Method
  - Request-URI
  - HTTP Version
  - Header Fields and MIME Types

Header Fields and MIME Types

- Example HTTP Request

```
POST /servlet/EchoHttpRequest HTTP/1.1
host: www.example.org:56789
user-agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.4)
  Gecko/20030624
accept: text/xml,application/xml,application/xhtml+xml,
  text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,
  image/gif;q=0.2,*/*;q=0.1
accept-language: en-us,en;q=0.5
accept-encoding: gzip,deflate
accept-charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
connection: keep-alive
keep-alive: 300
content-type: application/x-www-form-urlencoded
content-length: 13
```

HTTP Request Header Fields

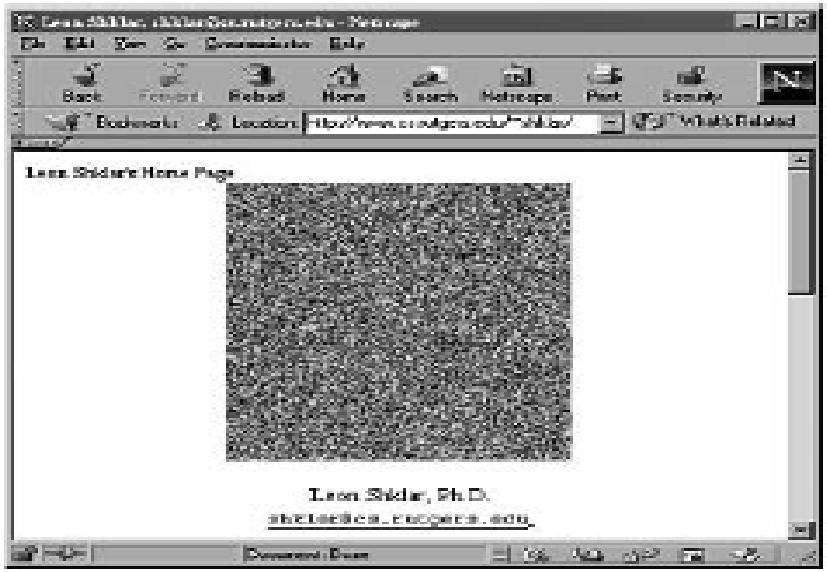
Field Name	Use
Host	Specify *authority* portion of URL (host plus port number; see Section 1.6.2). Used to support *virtual hosting* (running separate web servers for multiple fully qualified domain names sharing a single IP address).
User-Agent	A string identifying the browser or other software that is sending the request.
Accept	MIME types of documents that are acceptable as the body of the response, possibly with indication of preference ranking. If the server can return a document according to one of several formats, it should use a format that has the highest possible preference rating in this header.
Accept-Language	Specifies preferred language(s) for the response body. A server may have several translations of a document, and among these should return the one that has the highest preference rating in this header field. For complete information on registered language tags, see [RFC-3066] and [ISO-639-2].
Accept-Encoding	Specifies preferred encoding(s) for the response body. For example, if a server wishes to send a compressed document (to reduce transmission time), it may only use one of the types of compression specified in this header field.
Accept-Charset	Allows the client to express preferences to a server that can return a document using various character sets (see Section 1.5.4).

HTTP Request Header Fields

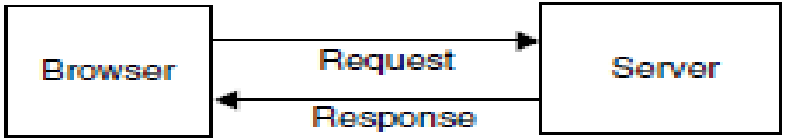
Connection	Indicates whether or not the client would like the TCP connection kept open after the response is sent. Typical values are `keep-alive` if connection should be kept open (the default behavior for servers/clients compatible with HTTP/1.1), and `close` if not.
Keep-Alive	Number of seconds TCP connection should be kept open.
Content-Type	The MIME type of the document contained in the message body, if one is present. If this field is present in a request message, it normally has the value shown in the example, application/x-www-form-urlencoded.
Content-Length	Number of bytes of data in the message body, if one is present.
Referer	The URI of the resource from which the browser obtained the Request-URI value for this HTTP request. For example, if the user clicks on a hyperlink in a web page, causing an HTTP request to be sent to a server, the URI of the web page containing the hyperlink will be sent in the Referer field of the request. This field is not present if the HTTP request was generated by the user entering a URI in the browser's Location bar.

Sequence of Requests and Responses to Load a Simple Web Page

Step 1: Initial user request for "`http://www.cs.rutgers.edu/~shklar/`"



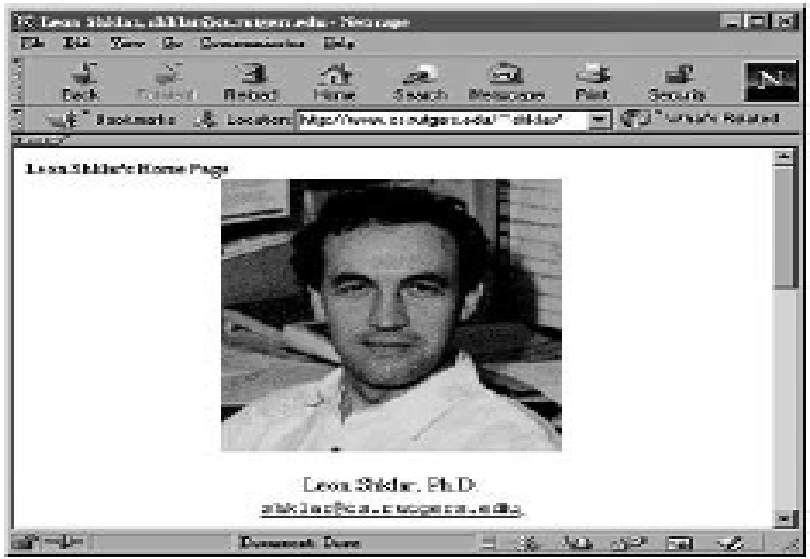

```
GET /~shklar/ HTTP/1.1
Host: www.cs.rutgers.edu
```



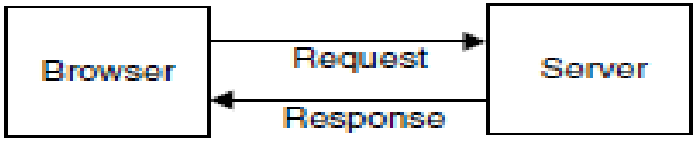

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<head> ... </head>
<body>
...
<img src="./images/photo.gif">
...
</body>
</html>
```

Sequence of Requests and Responses to Load a Simple Web Page

Step 2: Secondary browser request for `"http://www.cs.rutgers.edu/~shklar/images/photo.gif"`



```
GET /~shklar/images/photo.gif HTTP/1.1
Host: www.cs.rutgers.edu
```




```
HTTP/1.1 200 OK
Content-Type: image/gif
```

GET vs POST

- Applications are constructed so that it knows to
  - display a form when it receives a request using the GET method
  - process the form (and to present the results of processing the form) when it receives a request using the POST method

HTTP Response Message

- Response Status Line
- Response Header Fields
- Cache Control
- Character Sets

HTTP Response Message

- HTTP response message consists of a status line, header fields, and the body of the response, in the following format:
  - Status line
  - Header field(s) (one or more)
  - Blank line
  - Message body (optional)

HTTP Response Message

- Status line
  - HTTP/1.1 200 OK
- three fields:
  - the HTTP version used by the server software when formatting the response;
  - a numeric *status code* indicating the type of response;
  - a text string (the *reason phrase) that presents the* information represented by the numeric status code in human-readable form.
- This particular status code indicates that no errors were detected by the server.
- The body of a response having this status code should contain the resource requested by the client.
- All status codes are three-digit decimal numbers

HTTP/1.1 Status Code Classes

Digit	Class	Standard Use
1	Informational	Provides information to client before request processing has been completed.
2	Success	Request has been successfully processed.
3	Redirection	Client needs to use a different resource to fulfill request.
4	Client Error	Client's request is not valid.
5	Server Error	An error occurred during server processing of a valid client request.

HTTP/1.1 Status Codes

Status Code	Recommended Reason Phrase	Usual Meaning
200	OK	Request processed normally.
301	Moved Permanently	URI for the requested resource has changed. All future requests should be made to URI contained in the Location header field of the response. Most browsers will automatically send a second request to the new URI and display the second response.
307	Temporary Redirect	URI for the requested resource has changed at least temporarily. This request should be fulfilled by making a second request to URI contained in the Location header field of the response. Most browsers will automatically send a second request to the new URI and display the second response.
401	Unauthorized	The resource is password protected, and the user has not yet supplied a valid password.
403	Forbidden	The resource is present on the server but is read protected (often an error on the part of the server administrator, but may be intentional).
404	Not Found	No resource corresponding to the given Request-URI was found at this server.
500	Internal Server Error	Server software detected an internal failure.

Response Header Fields

Field Name	Use
Date	Time at which response was generated. Used for cache control (see Section 1.5.3). This field must be supplied by the server.
Server	Information identifying the server software generating this response.
Last-Modified	Time at which the resource returned by this request was last modified. Can be used to determine whether cached copy of a resource is valid or not (see Section 1.5.3).
Expires	Time after which the client should check with the server before retrieving the returned resource from the client's cache (see Section 1.5.3).
ETag	A hash code of the resource returned. If the resource remains unchanged on subsequent requests, then the ETag value will also remain unchanged; otherwise, the ETag value will change. Used for cache control (see Section 1.5.3).
Accept-Ranges	Clients can request that only a portion (*range*) of a resource be returned by using the Range header field. This might be used if the resource is, say, a large PDF file and only a single page is currently needed. Accept-Ranges specifies the units that may be used by the client in a range request, or none if range requests are not accepted by this server for this resource.
Location	Used in responses with redirect status code to specify new URI for the requested resource.

Cache Control

- HTTP caching
  - Allows HTTP responses to be held in temp storage to improve performance
  - Eliminates overhead of re-executing original request and improves server throughput
  - Types
    - Server-side
    - Browser-side
    - Proxy-side
      - In real world, intermediate proxies receive client requests, relay msgs through firewalls

Cache Control

- Caching rules
  - based on the Cache-Control header.
  - Valid settings include public, private, and no-cache.
  - The **public** setting removes all restrictions and authorizes both shared and non-shared caching mechanisms to cache the response.
  - The **private** setting indicates that the response is directed at a single user and should not be stored in a shared cache.
  - The **no-cache** setting indicates that neither browsers nor proxies are allowed to cache the response.

Cache Control

```
HTTP/1.1 200 OK
Date: Mon, 05 Feb 2001 03:26:18 GMT
Server: Apache/1.2.5
Last-Modified: Mon, 05 Feb 2001 03:25:36 GMT
Cache-Control: private
Pragma: no-cache
Content-Length: 2255
Content-Type: text/html

<html>
  ...
</html>
```

Cache Control

- HTTP caching
  - the image can be retrieved from the client file system rather than sending another HTTP request to the server and waiting for the server's response containing the image.
  - leads to quicker display by the browser, reduced network communication, and reduced load on the web server.

Cache Control

- Is cache valid?
  - Ask the server by sending an HTTP request for the resource using the HEAD method.
    - If Last-Modified time < Date header field value, then the cached copy is still valid
  - The client can compare the ETag returned by a HEAD request with the ETag stored with the cached resource. If the ETag values match, then the cached copy is valid.
  - the server can return time in an Expires header. In this case, as long as the Expires time has not been reached, the client may use the cached copy of the resource without the need to validate with the server.

HTTP Authentication

- Authentication
  - Verifying user id
- Authorization
  - Check whether user has access to a resource
- Basic authentication
  - User id and pass are transmitted via Authorization header as a single encoded string
  - Pop-up authentication dialog, then web app is using built-in HTTP authentication

HTTP Authentication

```
HTTP/1.1 401 Authenticate
Date: Mon, 05 Feb 2001 03:41:23 GMT
Server: Apache/1.2.5
WWW-Authenticate: Basic realm="Chapter3"
```

```
GET /book/chapter3/index.html HTTP/1.1
Date: Mon, 05 Feb 2001 03:41:24 GMT
Host: www.neurozen.com
Authorization: Basic eNCoDEd-uSErId:pASswORd
```

Session support through Cookie and Set-Cookie headers

- Add items to your ‘cart’ as you decide to purchase them.
- When the time comes to process your order, the site seems to remember what items you have placed in your cart.
- But how does it know this, if HTTP requests are atomic and disconnected from each other?

Session support through Cookie and Set-Cookie headers

- Set-Cookie
  - a response header sent by the server to the browser, setting attributes that establish state within the browser.
- Cookie
  - a request header transmitted by the browser in subsequent requests to the same (or related) server. It helps to associate requests with *sessions*

Session support through Cookie and Set-Cookie headers

```
Set-Cookie:  <name>=<value>[;  expires=<date>][;  path=<path>]
          [; domain=<domain name>][; secure]
```

```
HTTP/1.1 200 OK
Set-Cookie: name=Leon; path=/test/; domain=.rutgers.edu
```

Session support through Cookie and Set-Cookie headers

- Movie rental example

```
GET /movies/register HTTP/1.1
Host: www.sample-movie-rental.com
Authorization:...
```

Once the server has recognized and authenticated the user, it sends back a response containing a `Set-Cookie` header containing a client identifier:

```
HTTP/1.1 200 OK
Set-Cookie: CLIENT=Rich; path=/movies
...
```

Session support through Cookie and Set-Cookie headers

- Movie rental example

```
GET /movies/rent-recommended HTTP/1.1
Host: www.sample-movie-rental.com
Cookie: CLIENT=Rich
```

In this case, we are visiting a recommended movie page. The server response now contains a movie recommendation:

```
HTTP/1.1 200 OK
Set-Cookie: MOVIE=Matrix; path=/movies/
...
```

Session support through Cookie and Set-Cookie headers

- Movie rental example

```
GET /movies/access HTTP/1.1
Host: www.sample-movie-rental.com
Cookie: CLIENT=Rich; MOVIE=Matrix
```

We get back the acknowledgement containing access information to the recommended movie for future status checks:

```
HTTP/1.1 200 OK
Set-Cookie: CHANNEL=42; PASSWD=Matrix007; path=/movies/status/
...
```

Web Clients

- Browser functions
- URLs
- User Controllable Features

Web Clients

- A *web client is software that accesses a web server by sending an HTTP request message* and processing the resulting HTTP response.
- any web client that is designed to directly support user access to web servers is known as a *user agent.*
- *Mosaic 1993*
- *Netscape and Internet Explorer browser war*

Web Clients

- A primary task of any browser is to make HTTP requests on behalf of the browser user.
- If a user types an http-scheme URL in the Location bar, for example, the browser must perform a number of tasks:
  - Reformat the URL entered as a valid HTTP request message.
  - If the server is specified using a host name (rather than an IP address), use DNS to convert this name to the appropriate IP address.
  - Establish a TCP connection using the IP address of the specified web server.
  - Send the HTTP request over the TCP connection and wait for the server's response.
  - Display the document contained in the response. If the document is not a plain-text document but instead is written in a language such as HTML, this involves *rendering* the document: positioning text and graphics appropriately within the browser window, creating table borders, using appropriate fonts and colors, etc.

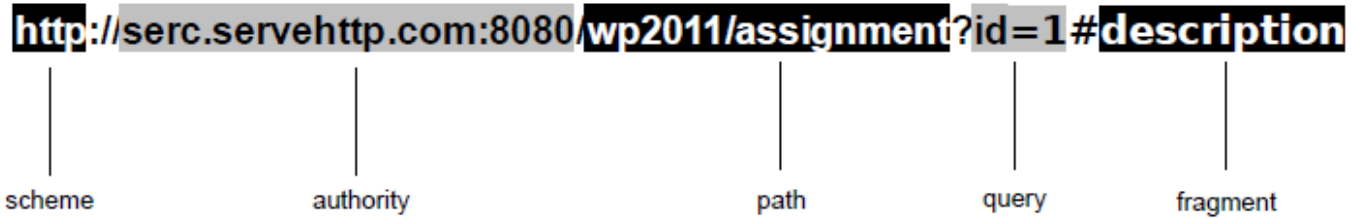
URLs

```
scheme://host[:port#]/path/../[;url-params][?query-string][#anchor]
```

URLs

- http://www.example.org:56789/a/b/c.txt?t=win&s=chess#para5
  - **Authority**: is www.example.org:56789 and consists of the fully qualified domain name www.example.org followed by the port number 56789.
  - **Path** is /a/b/c.txt where / refers to root
  - information between ?and # is the ***query string,*** intended to pass search terms to a web server
  - ***Fragment*** *identifiers are used by browsers* to scroll HTML documents
  - Corresponding browser request will be
    - GET /a/b/c.txt?t=win&s=chess HTTP/1.1
    - ...
    - Host: www.example.org:56789
    - ...

URLs



- **Scheme*:** identifies syntax & semantics for URIs using that scheme e.g. http tells agent to use http protocol
- **Authority:** identifies the server address and port. It may also contain optional user info followed by "@" symbol.
- **Path*:** represents the path of resource on the server.
- **Query:** additional information to identify the resource in conjunction with Path
- **Fragment:** represents a sub-resource within the given resource identified by Path & Query information e.g. a para within a page

Web Servers

This NeXT Computer was used by Tim Berners-Lee at CERN and became the world's first Web server.

Web Servers

- Web server's job is to:
  - host web content (static / dynamic)
  - Support technologies for efficient creation, deployment and distribution of content
  - Support multiple concurrent user requests
  - Support secure communication
  - Support multiple websites / applications simultaneously
- Modern Web servers
  - Apache
  - IIS
  - …

Web Servers

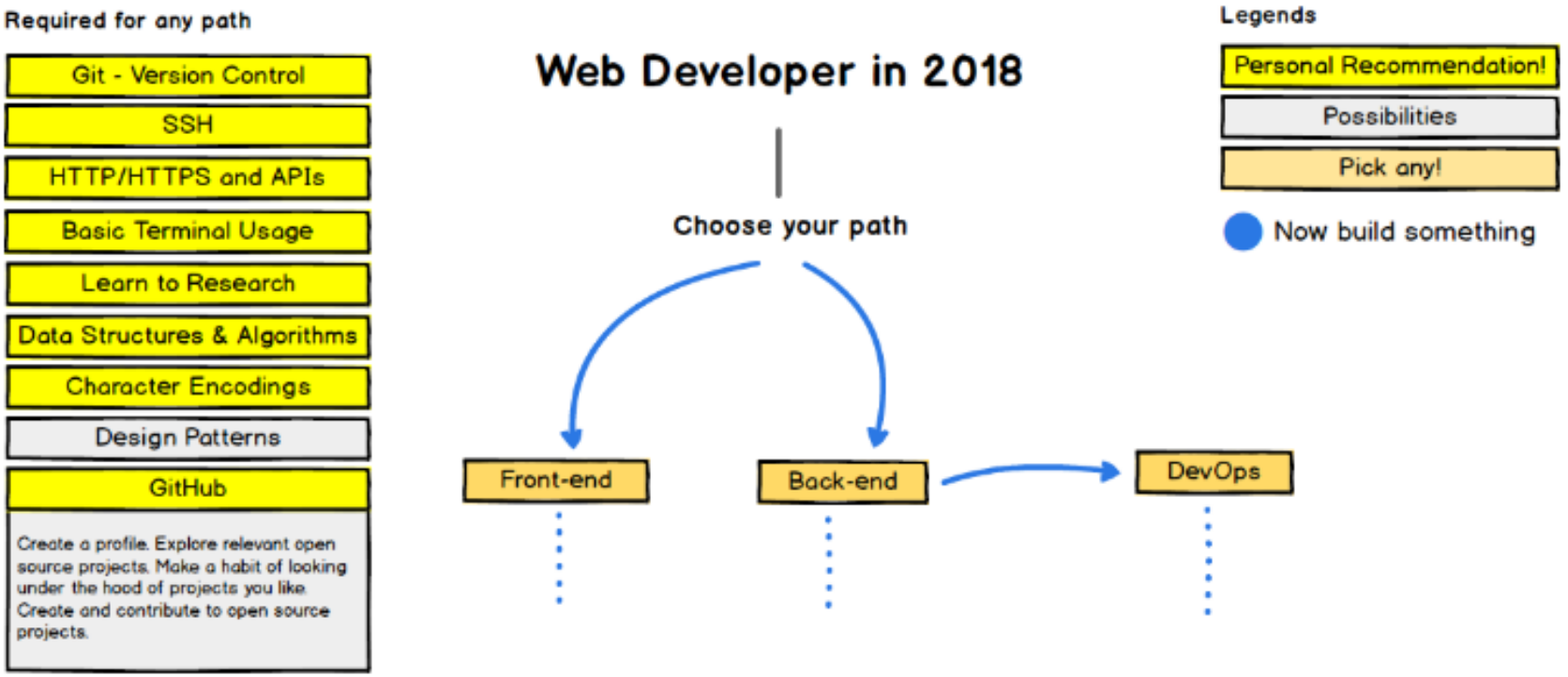
- Accept HTTP requests from web clients and return an appropriate resource (if available) in the HTTP response
  1. The server calls on TCP software and waits for connection requests to one or more ports.
  2. When a connection request is received, the server establishes the TCP connection and receives an HTTP request.
  3. maps the Request-URI field of the HTTP request start line to a resource on the server.

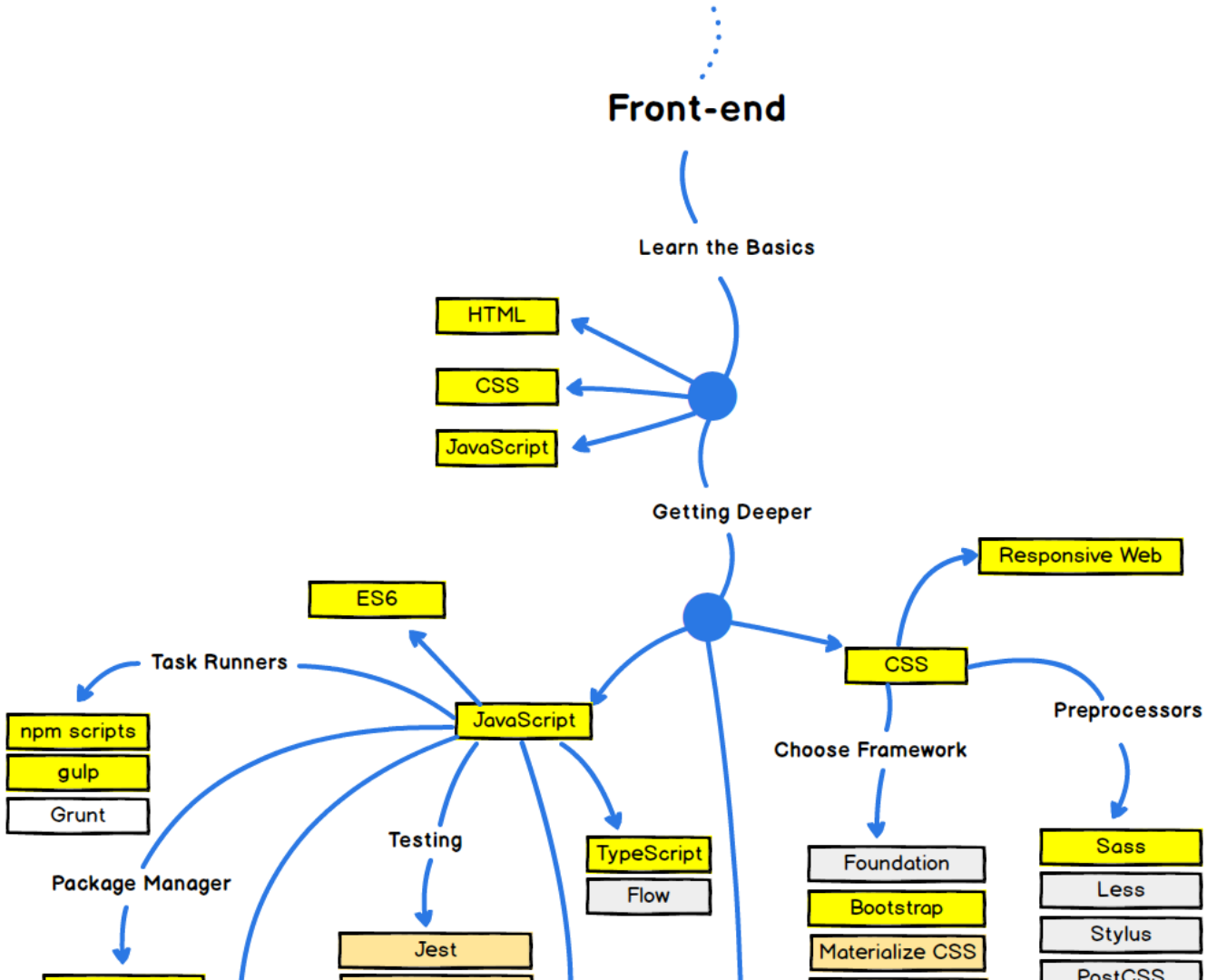
Web Servers

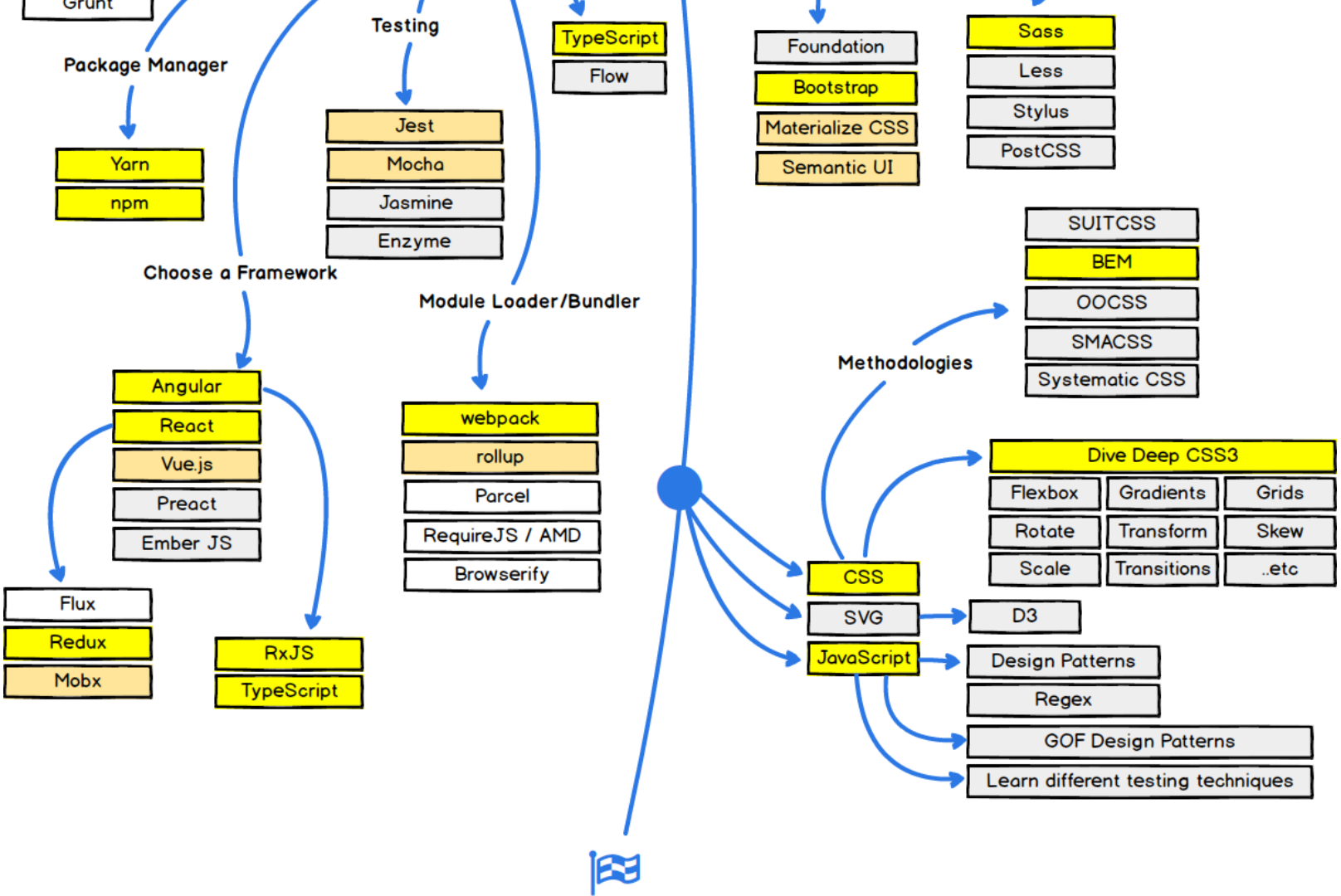
- Server Features
  4. If the resource is a file, the host software determines the MIME type of the file (usually by a mapping from the file-name extension portion of the Request-URI), and creates an HTTP response that contains the file in the body of the response message.
  5. If the resource is a program, the host software runs the program, providing it with information from the request and returning the output from the program as the body of an HTTP response message.
  6. The server normally logs information about the request and response—such as the IP address of the requester and the status code of the response—in a plain-text file.
  7. If the TCP connection is kept alive, the server continues to monitor the connection until a certain length of time has elapsed, the client sends another request, or the client initiates a connection close.

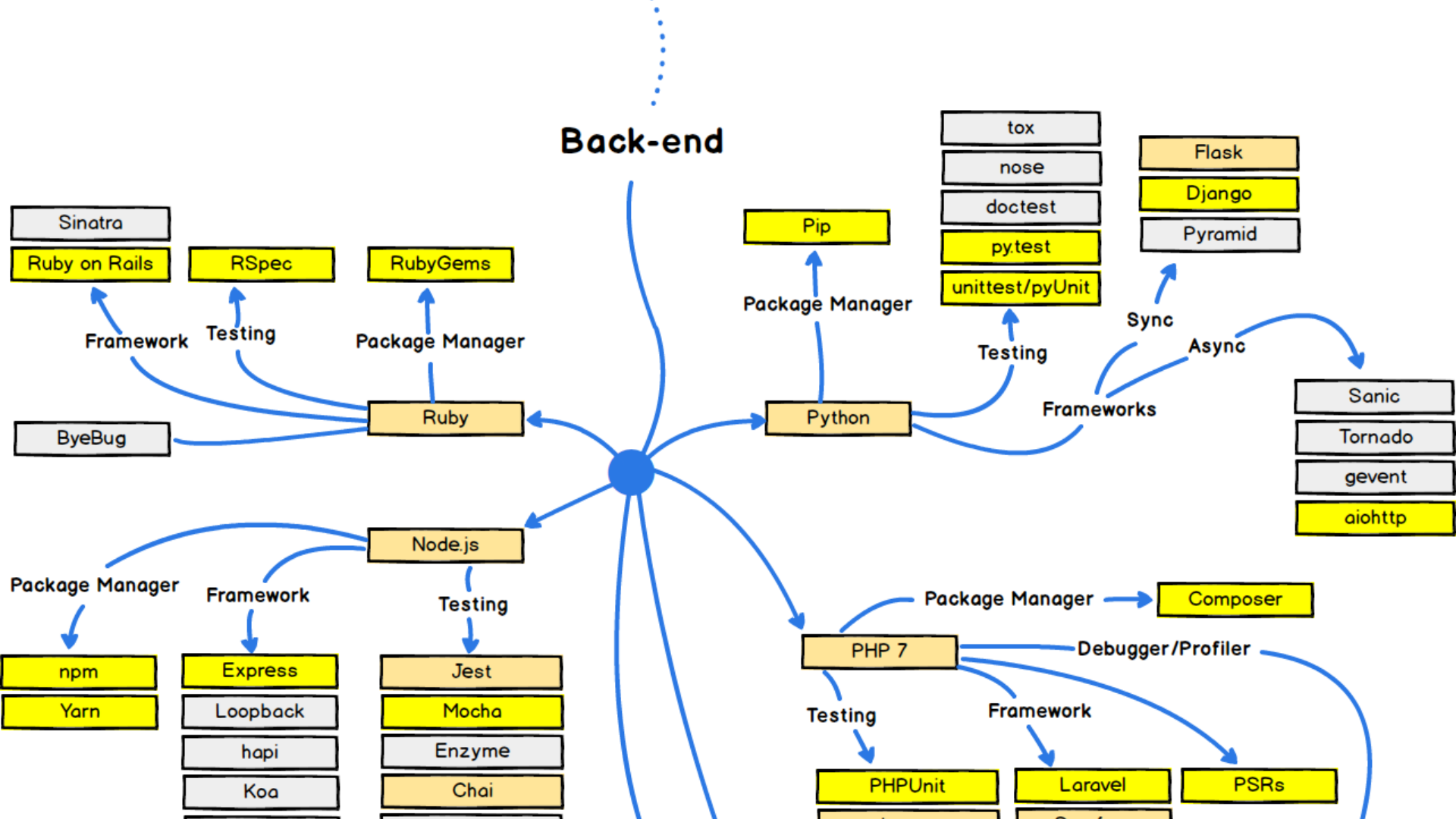
Web Technologies

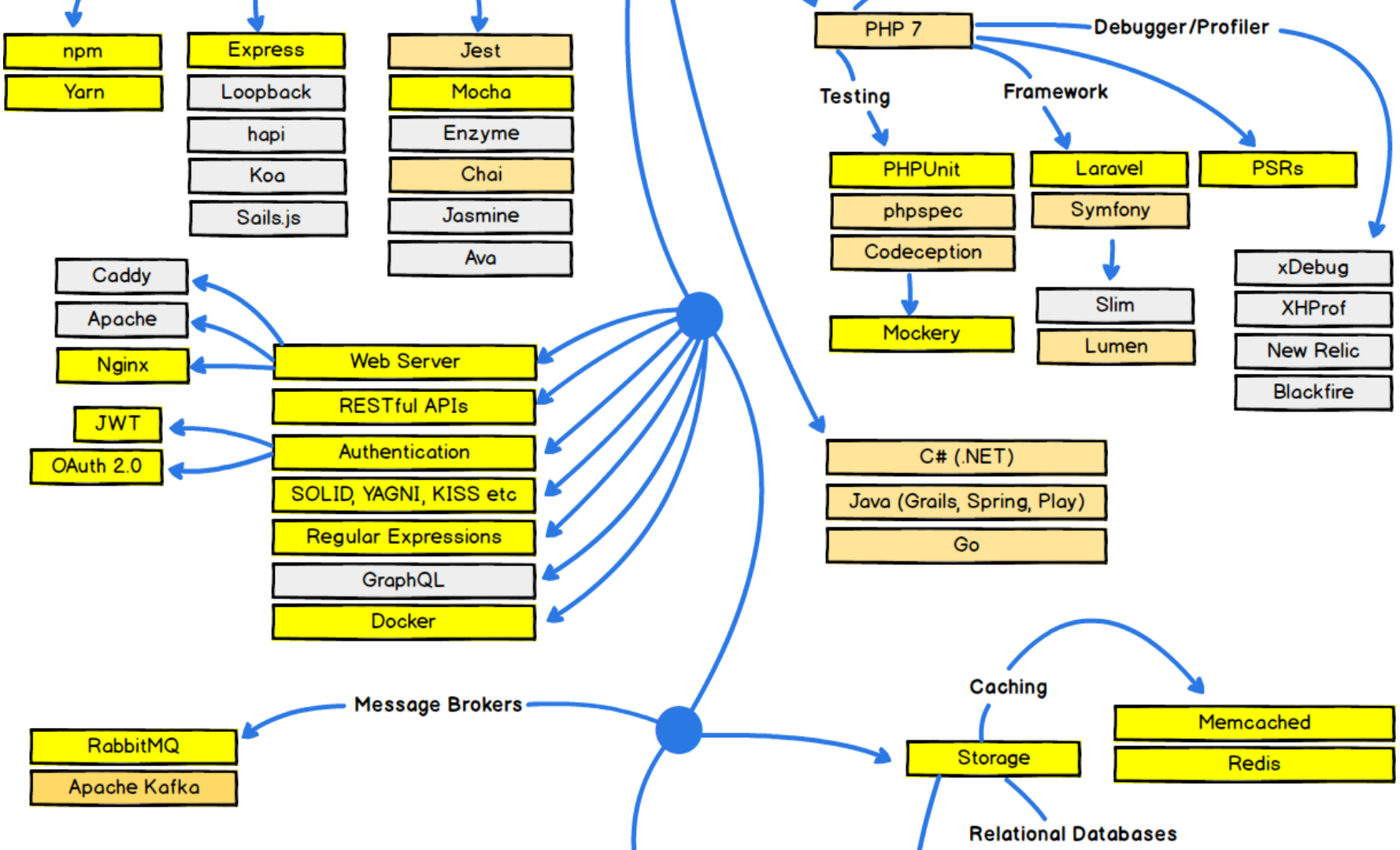
Web Developer in 2018

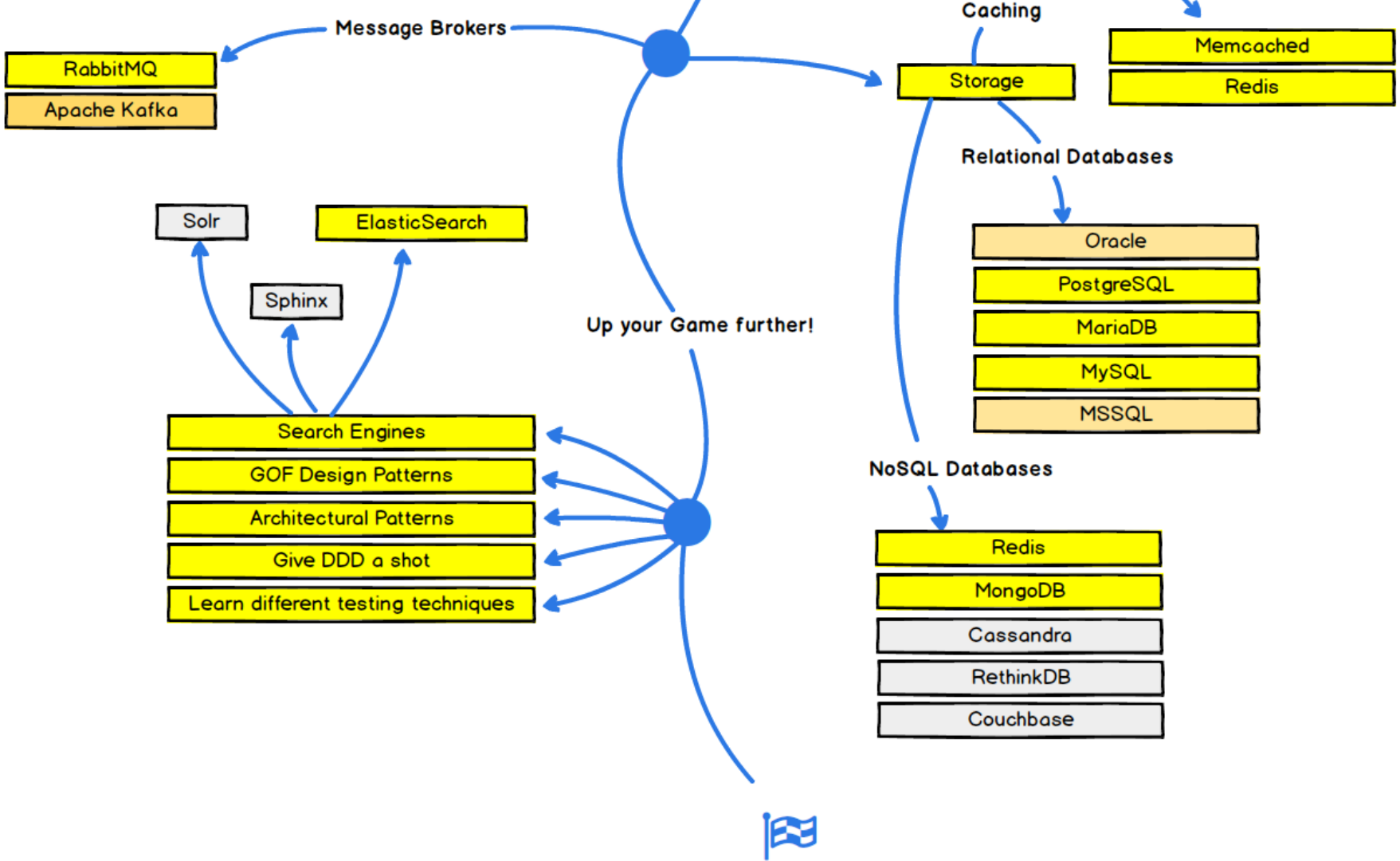



Front-end
Learn the Basics
HTML
CSS
JavaScript
Getting Deeper
ES6
Task Runners
npm scripts
gulp
Grunt
JavaScript
Responsive Web
CSS
Preprocessors
Choose Framework
Package Manager
Testing
Jest
TypeScript
Flow
Foundation
Bootstrap
Materialize CSS
Sass
Less
Stylus
PostCSS


Grunt
Package Manager
Yarn
npm
Testing
Jest
Mocha
Jasmine
Enzyme
TypeScript
Flow
Foundation
Bootstrap
Materialize CSS
Semantic UI
Sass
Less
Stylus
PostCSS
Choose a Framework
Angular
React
Vue.js
Preact
Ember JS
Flux
Redux
Mobx
RxJS
TypeScript
Module Loader/Bundler
webpack
rollup
Parcel
RequireJS / AMD
Browserify
Methodologies
SUITCSS
BEM
OOCSS
SMACSS
Systematic CSS
CSS
Dive Deep CSS3
Flexbox
Gradients
Grids
Rotate
Transform
Skew
Scale
Transitions
..etc
SVG
D3
JavaScript
Design Patterns
Regex
GOF Design Patterns
Learn different testing techniques


Back-end
Sinatra
Ruby on Rails
RSpec
RubyGems
Framework
Testing
Package Manager
Ruby
ByeBug
Pip
Package Manager
Python
tox
nose
doctest
py.test
unittest/pyUnit
Testing
Flask
Django
Pyramid
Sync
Async
Frameworks
Sanic
Tornado
gevent
aiohttp
Node.js
Package Manager
Framework
Testing
npm
Yarn
Express
Loopback
hapi
Koa
Jest
Mocha
Enzyme
Chai
PHP 7
Package Manager
Composer
Debugger/Profiler
Testing
Framework
PHPUnit
Laravel
PSRs


npm
Yarn
Express
Loopback
hapi
Koa
Sails.js
Jest
Mocha
Enzyme
Chai
Jasmine
Ava
PHP 7
Debugger/Profiler
Testing
Framework
PHPUnit
phpspec
Codeception
Mockery
Laravel
Symfony
Slim
Lumen
PSRs
xDebug
XHProf
New Relic
Blackfire
Caddy
Apache
Nginx
Web Server
RESTful APIs
JWT
OAuth 2.0
Authentication
SOLID, YAGNI, KISS etc
Regular Expressions
GraphQL
Docker
C# (.NET)
Java (Grails, Spring, Play)
Go
Message Brokers
RabbitMQ
Apache Kafka
Caching
Storage
Memcached
Redis
Relational Databases


Message Brokers
RabbitMQ
Apache Kafka
Caching
Memcached
Redis
Storage
Relational Databases
Oracle
PostgreSQL
MariaDB
MySQL
MSSQL
NoSQL Databases
Redis
MongoDB
Cassandra
RethinkDB
Couchbase
Solr
ElasticSearch
Sphinx
Search Engines
GOF Design Patterns
Architectural Patterns
Give DDD a shot
Learn different testing techniques
Up your Game further!

Web Application Architecture

Web Application Architecture

- Client server architecture
- MULTI-TIER (2-TIER, 3-TIER)
- MODEL-VIEWER-CONTROLLER (MVC)

Web Development Concerns

- Understand the intricacies of web architecture
- Understand **technologies** (client and server side both) that support efficient creation and consumption of web content for quality user experience
- Build **architectures** for integration of web-based applications with other aspects of underlying business for supporting diverse functions and allow extensibility in case of changing business needs
- Address **performance** and scalability issues
- Ensure **security** of communication over web

REST

- Representational state transfer
  - an architectural style that defines a set of constraints to be used for creating web services.
  - Web Services that conform to the REST architectural style are **RESTful**
  - In a RESTful web service, requests made to a resource's URI will get a response with a payload formatted in either HTML, XML, JSON, or some other format.

REST

- Representational state transfer
  - The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.
  - Presented an image of how a well-designed Web application behaves:
    - it is a network of Web resources (a virtual state-machine)
    - where the user progresses through the application by selecting links, such as /user/tom,
    - and operations such as GET or DELETE (state transitions),
    - resulting in the next resource (representing the next state of the application) being transferred to the user for their use.

REST Architectural constraints

- Client–server architecture
- Statelessness
- Cacheability
- Layered system
- Code on Demand
- Uniform Interface

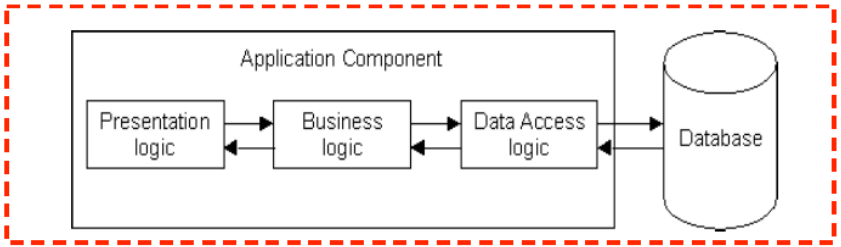
REST Architectural constraints

- An application or architecture considered RESTful or REST-style is characterized by:
  - State and functionality are divided into distributed resources
  - Every resource is uniquely addressable using a uniform and minimal set of commands (typically using HTTP commands of GET, POST, PUT, or DELETE over the Internet)
  - The protocol is client/server, stateless, layered, and supports caching
- This is essentially the architecture of the Internet and helps to explain the popularity and ease-of-use for REST.

REST and Web Services

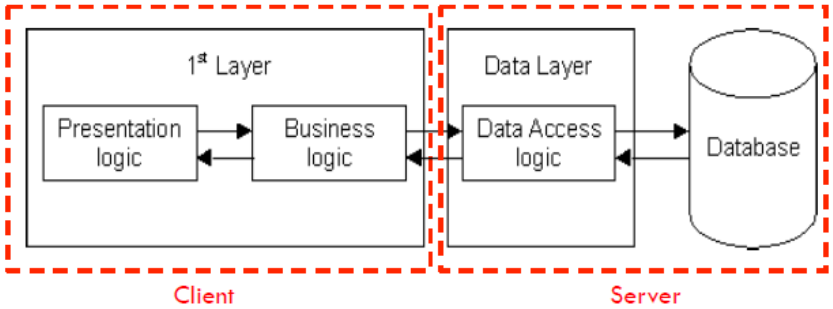
1-Tier Architecture

- All 3 layers are on the same machine
- All code and processing kept on a single machine
- Presentation, Logic, Data layers are tightly connected



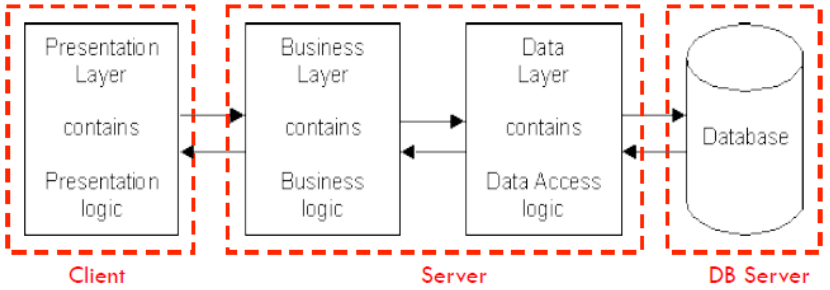
2-Tier Architecture

- Database runs on Server
  - Separated from client
  - Easy to switch to a different database
- Presentation and logic layers still tightly connected
  - Heavy load on server
  - Potential congestion on network
  - Presentation still tied to business logic

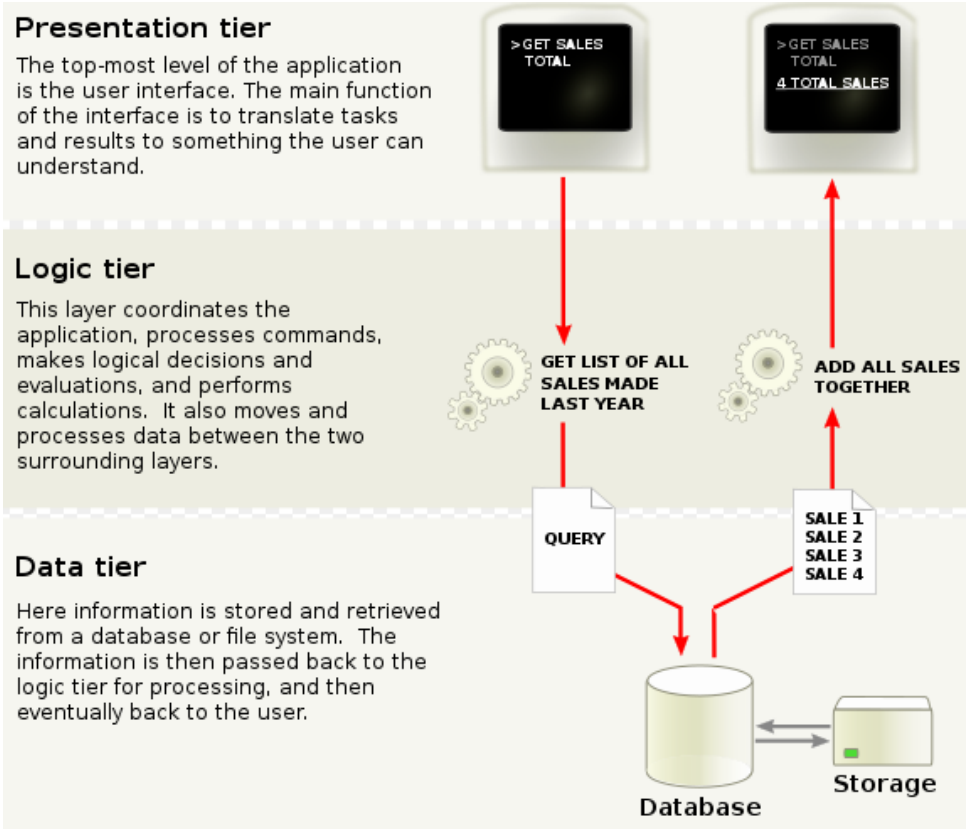


3-Tier Architecture

- Each layer can potentially run on a different machine
- Presentation, logic, data layers disconnected



Example 3-Tier Application



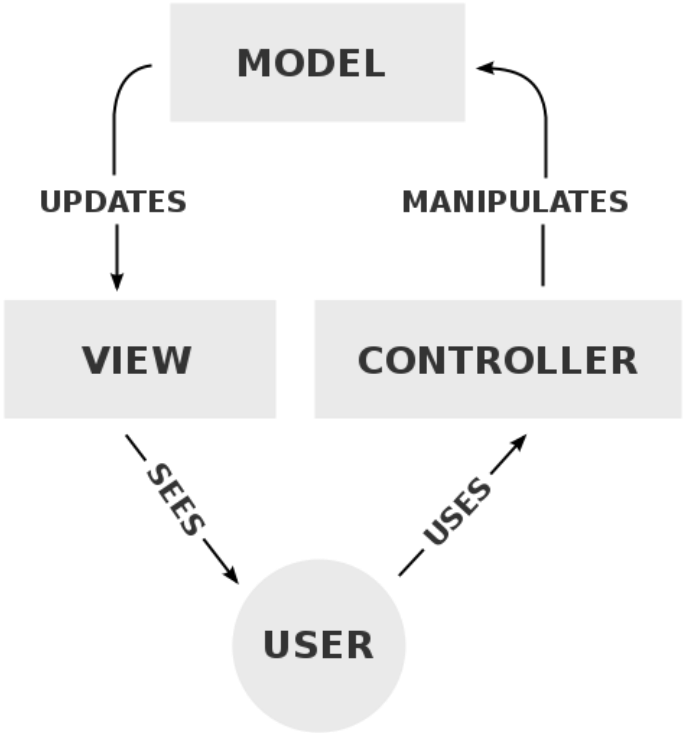
The 3-Tier Architecture for Web Apps

- Presentation Layer
  - Static or dynamically generated content rendered by the browser (front-end)
- Logic Layer
  - A dynamic content processing application server, e.g., ASP.NET, PHP, ColdFusion platform (middleware)
- Data Layer
  - A database, manages and provides access to the data (back-end)

3-Tier Architecture - Advantages

- Independence of Layers
  - Easier to maintain
  - Components are reusable
- Faster development (division of work)
  - Web designer does presentation
  - Software engineer does logic
  - DB admin does data model

MVC Pattern



MVC

- Model
  - Contains domain-specific knowledge
  - Records the state of the application
  - E.g., what items are in a shopping cart
  - Often linked to a database
  - Independent of view
  - One model can link to different views
- View
  - Presents data to the user
  - Allows user interaction
  - Does no processing
- Controller
  - defines how user interface reacts to user input (events)
  - receives messages from view (where events come from)
  - sends messages to model (tells what data to display)

MVC for Web Applications

- Model
  - database tables (persistent data)
  - session information (current system state data)
- View
  - (X)HTML
  - CSS style sheets
- Controller
  - client-side scripting
  - http request processing
  - business logic/preprocessing

Recap

Questions?

Suggestions?

One Devloper Army
@OneDevloperArmy

If you are a new programmer I just want you to know

me and all of my colleagues with years of experience

Google the most basic things _daily_

12:18 PM · 13 Aug 18

View Tweet activity

Homework

- Make your GitHub account
- The link for the github classroom assignment is https://classroom.github.com/a/VyEshKn1 and the instructions for the assignment are in the readme file.
- When you accept the assignment a repository will be created in your Github account with the practice prefix. E.g. practice-*yourusername*
- *You have to push a text file (containing your name and roll number) to this repository.*
- *The purpose of this assignment is to have some Git practice*

Next

- HTML

References

- A: Web Application Architecture Principles, Protocols and Practices (Leon Shklar and Rich Rosen)
- B: Web Technologies (Jeffrey Jackson)
- Wikipedia
- https://www.codementor.io/brandonmorelli/the-2018-web-developer-roadmap-g9dte7x61
- https://github.com/kamranahmedse/developer-roadmap