Seneca
SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

WEB322

Web Programming Tools and Frameworks

Schedule
Notes
Graded Work
Resources
Heroku Guide
MyApps Instructions
Code examples

WEB322 Week 1 Notes

Introduction

Welcome to **WEB322 – Web Programming Tools and Frameworks.** In this course we will be studying a wide range of technologies that are used to create dynamic content on the web. These include modern tools and libraries / frameworks that enable the programmer to quickly and efficiently create a functioning web-based application capable of responding to requests for content, reacting predictably to errors and storing / retrieving user and application data. We will be looking at what exactly a web application is and how we can use a familiar programming language (JavaScript / ECMAScript) to create and maintain one. Additionally, we will be studying how web browsers send data to and from a web server and how we can ensure that our applications are scalable, secure and robust. We will also study methods of storing and retrieving data from a data store (SQL & NoSQL Databases) and how to manage state (ie: “logged-in”) information about users.

Development Environments

Throughout this course, we will be working almost exclusively in the following environments:

Visual Studio Code

Visual Studio

“Visual Studio Code is an open-source (free) streamlined code editor with support for development operations like debugging, task running and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs”. Visual Studio Code also runs on Mac OS X, Linux and Windows operating systems, which will provide the class with a single unified environment to work in regardless of a student’s choice of laptop or home computer. Some of the noteworthy features of Visual Studio Code Include:

Integrated Terminal

“In Visual Studio Code, you can open an integrated terminal, initially starting at the root of your workspace. This can be very convenient as you don’t have to switch windows or alter the state of an existing terminal to perform a quick command line task”.

To open the terminal:

- Use the keyboard shortcut **Ctrl + `**
- Use the **View** | **Toggle Integrated Terminal menu command.**

Smart Editing

VS Code comes with a built-in JavaScript language service so you get JavaScript code intelligence out-of-the-box. Language services provide the code understanding necessary for features like:

- IntelliSense: (suggestions)
- smart code navigation (Go to Definition, Find All References, Rename Symbol)

File & Folder Based

Since VS Code is file and folder based – you can get started immediately by simply opening a file or folder in VS Code.

"On top of this, VS Code can read and take advantage of a variety of project files defined by different frameworks and platforms. For example, if the folder you opened in VS Code contains one or more package.json (which we will be making extensive use of during the semester), project.json, tsconfig.json, or .NET Core Visual Studio solution and project files, VS Code will read these files and use them to provide additional functionality, such as rich IntelliSense in the editor".

Version Control

Visual Studio Code has integrated Git support for the most common commands. Git is essentially an open-source "Version Control" System, often used by developers in a team setting (GitHub) to manage updates to their code in a centralized location, while also maintaining a history of changes (versions). While we will not be using Git or GitHub in this class, Git is an extremely popular Version Control System used for both commercial and open-source projects around the world and having integrated access to the commands from within our IDE is a valuable feature. In higher level courses, having this feature will make it more straightforward to manage your code commits while you develop.

Modern Web Browser

chrome Firefox

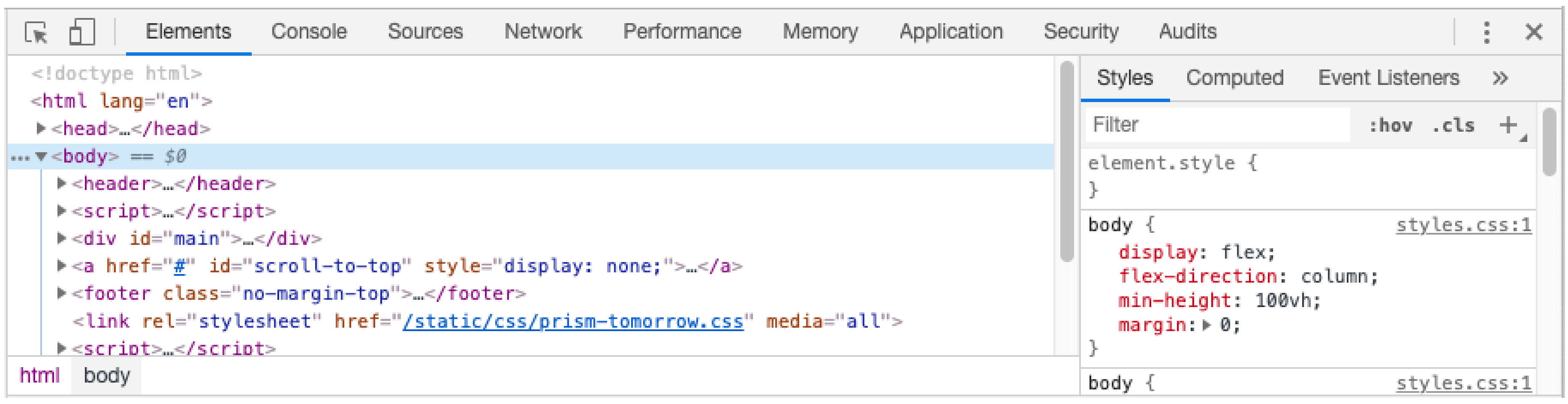
A modern web browser such as Google Chrome or Mozilla Firefox will be used regularly throughout this course. Internet Explorer 11 / Edge will work as well, as it supports a similar set of development tools, however due to it's lack of plugins / addons and cross-platform support it's not as highly recommended. All screenshots and development examples used throughout this course have been taken in Google Chrome.

Browser Developer Toolbar

Before starting this course, students should have at least a basic understanding of the Developer Tools built into a modern web browser. Typically, pressing the F12 Key (Windows) will open the bar, however there are alternate ways of opening it. For Google chrome:

- Open the Chrome menu at the top-right of your browser window, then select Tools > Developer Tools.
- Right-click on any page element and select Inspect.

This will bring up the Chrome "Developer Toolbar", as seen below :



```
<!doctype html>
<html lang="en">
 ▶<head>…</head>
…▼<body> == $0
   ▶<header>…</header>
   ▶<script>…</script>
   ▶<div id="main">…</div>
   ▶<a href="#" id="scroll-to-top" style="display: none;">…</a>
   ▶<footer class="no-margin-top">…</footer>
    <link rel="stylesheet" href="/static/css/prism-tomorrow.css" media="all">
   ▶<script>…</script>
```

```
element.style {
}
body {                                   styles.css:1
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  margin:▶ 0;
}
body {                                   styles.css:1
```

We will be working with many of these panels throughout the semester. A quick list of their functionality (from left to right, starting at the top left corner) is as follows:

Element Inspector: Select an element in the page to inspect it; this will cause the Developer Tools (Devtools) to switch to the "Elements" panel and highlight the rendered source code (HTML) responsible for displaying the item. This will also cause the "Styles" panel (on the right) to highlight all current CSS applied to the element

◫	**Device Toolbar Toggle:** Toggles the “device toolbar” on and off. This allows the developer to select a device and manually enter the pixel dimensions of the screen and scale of the page. This is useful for ensuring that the page looks correct in a variety of devices
Elements	**Elements Panel:** Shows a view of the current page’s Document Object Model (DOM) tree as HTML. Selecting a given node (element) will highlight it in the page and show it’s applied CSS in the “Styles” panel. Developers can also modify this element and corresponding CSS (“Styles” panel) live and see the results directly in the browser. Important Note: The HTML shown in this panel isn’t necessarily the source code of the page, as it will show elements and attributes that have been dynamically added after the page is loaded. Changes to the HTML/CSS/JavaScript in this mode will not save to the source file.
Console	**Console Panel:** shows a JavaScript console pane. JavaScript calls to “console.log()” will show the resultant text in this window. Additionally, all JavaScript errors will show up in this location in red. Developers can also write small JavaScript code snippets to be executed immediately within the context of the page.
Sources	**Sources Panel:** shows a list of all items included in the page (ie: all images, CSS, JavaScript, etc) and their corresponding locations of origin. Developers can click on an item to show it’s contents in the middle (preview) panel. If the selected item is a JavaScript file, developers can (in the “debugger” panel) set breakpoints and watch variables to help identify and debug a misbehaving piece of JavaScript code.
Network	**Network Panel:** is used to get additional insights into requested and downloaded resources. Developers can start/stop the recording of a log that tracks all resources loaded including their corresponding status code, type, time (latency), size of the resource and the initiator of the request
Performance	**Performance Panel:** enables a tool that allows developers to record and analyze all the activity in their applications as they run. It’s the best place to start investigating perceived performance issues. This is done by recording a timeline of every event that occurs after a page loads and analyzing the corresponding FPS, CPU, and network requests.
Memory	**Memory Panel:** provides more detailed debugging information than the timeline by enabling developers to record detailed CPU/Memory profiles such as a “JavaScript CPU Profile”, “Heap Snapshot”, “Allocating Timeline” and “Allocation Profile”.
Application	**Application Panel:** (previously, the “resource” panel) allows developers to inspect and manage client-side storage, caches, and resources. This includes: key-value pairs stored in “Local Storage”, access to IndexedDB Data (a JavaScript-based object-oriented database used to store data locally), a “Web SQL” explorer (depreciated in favour of IndexedDB), as well as access to stored cookies and cache data. This is very useful in verifying that your application is storing data correctly on the client side.
Security	**Security Panel:** gives an overview of a page from a security standpoint including: Certificate verification (indicating whether the site has proven its identity with a TLS certificate), Transport Layer Security (TLS) connection (Note: TLS is often referred to by the name of it’s predecessor, SSL) and Subresource security (indicating whether the site loads insecure HTTP subresources – ie: “mixed content”).
Audits	**Audits Panel:** is used to analyze a page as it loads. Once a page has finished loading, the audit provides suggestions and optimizations for decreasing page load time and increase perceived (and real) responsiveness. For example, it might suggest that a developer “Remove unused CSS rules” or “Combine external JavaScript” in an effort to optimize network utilization and page performance.
⊗ 1	**Error Icon:** displays the number of errors present in the “Console Pane”. To review the errors, simply switch over to the Console pane and locate the items highlighted in red.
⋮	**Customize Icon:** controls where the Developer Toolbar should be placed relative to the browser, as well as a collection of all related settings and preferences for the tool set.
×	**Close Icon:** closes the Developer Toolbar.

Core Technologies

Additionally, we will cover a number of topics surrounding the following technologies (in no particular order):

JavaScript (ES5 & ES6)

A huge focus of this course will be on JavaScript. In fact – JavaScript will be the only official programming language that we will be studying in this course. While we will be interacting with HTML5 and CSS3, neither is considered a "programming language" in the same way that C, C++ or JavaScript is. HTML5 and CSS3 are instead considered markup languages and style sheet languages respectfully – that is, they describe presentation, whereas programming languages describe function. Regardless, we will be focusing exclusively on JavaScript and how a number of very sophisticated tools and frameworks can help us create efficient and functional web applications.

ECMAScript

Back in 1996 the JavaScript language specification was taken to Ecma (European Computer Manufacturers Association) International to develop a formal standardized specification, which other browser vendors and companies could implement and expand. This standardized JavaScript was dubbed "ECMAScript" and specific vendor versions of the specification were known as "dialects", the most popular of which being "JavaScript". When we refer to "JavaScript" we're really referring to a dialect of ECMAScript that has been implemented in the engine / runtime environment that is running our JavaScript formatted code. For example, this includes JavaScript engines like SpiderMonkey in Firefox, v8 in Chrome, or Chakra in Internet Explorer.

In 2015, ECMAScript 6 was released and many interesting new features were introduced, such as:

- Arrow Functions
- Class Definitions
- Block Scoped Variables
- Promises
- Binary & Octal literals
- Modules
- and many more…

While not all ES6 features are fully implemented in all browsers (JavaScript engines / runtimes), the main server-side runtime that we will be working in (Node.js) supports 97% of the specification and is considered production ready.

Node.js

At it's core, Node.js is an open-source, cross-platform JavaScript runtime environment built on Chrome's V8 JavaScript engine. It is typically used for developing server-side and networking applications and has recently exploded as the go-to application framework for many real-time web applications. This is largely due to it's event-driven, non-blocking I/O model which ensures that the main thread of execution is not kept waiting for slow I/O operations (ie: stopping and waiting for a database query to complete). Some major companies using it include Paypal, eBay, GoDaddy, Heroku, Microsoft, Shutterstock, Uber, Wikia just to name a few.

Node.js also has an expansive package ecosystem accessible via it's Node Package Manager (NPM) utility. We will leverage this in WEB322 by experimenting with a number of popular, open-source modules including:

- Express.js (http://expressjs.com)
- Handlebars.js (http://handlebarsjs.com)
- Multer (https://github.com/expressjs/multer)
- Socket.io (http://socket.io)

jQuery

"jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript."

Back in 2006 when jQuery was initially released – we didn't have the same level of standardization that we do across browsers today. At the time, developers were desperately trying to get their client-side logic to work consistently across platforms such as Internet Explorer 7, Firefox 2.0, Safari 2, (Google Chrome still wasn't out yet) and many others. Where jQuery shone so brightly was it's robust selector engine and how it handled user events and AJAX calls consistently and painlessly, *regardless of the platform*. It sped up development and a huge community of developers began introducing "plugins" – and suddenly complex client-side operations (like building a fully-featured, drag-and-drop calendar) became fast and efficient. As a result, jQuery's popularity exploded and became the de facto JavaScript library for many developers and companies. A number of popular modern frameworks rely upon it, such as Bootstrap, Kendo UI and Foundation to name a few.

PostgreSQL

From the PostgreSQL site, postgresql.org:

> *"PostgreSQL (also known as "Postgres") is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, macOS, Solaris, Tru64), and Windows. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL:2008 data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, among others, and exceptional documentation.*

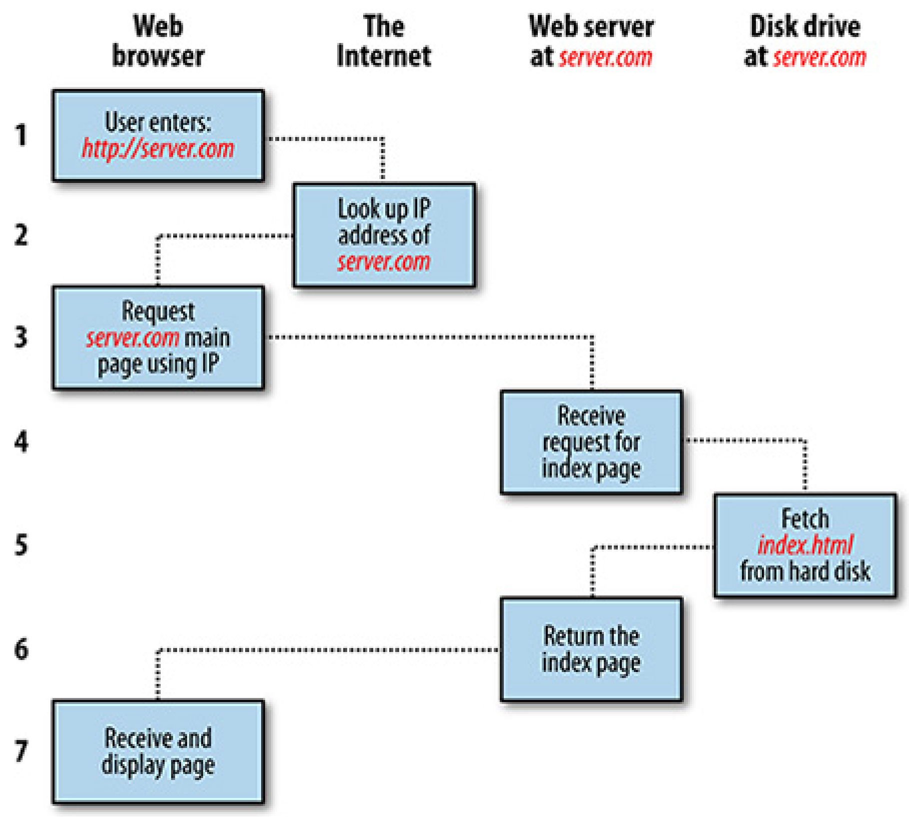
MongoDB

MongoDB is another open-source database that we will be exploring in this course. However, unlike MySQL MongoDB is classified as a “NoSQL” database and stores its data in JSON like format rather than in tables with fixed columns. The term NoSQL comes from “Not only SQL” and is intended to mean that it is a type of database system that can store data in non traditional tabular and relational format. It is because of this that NoSQL is quickly becoming a popular alternative to traditional Relational Databases (RDBMS).

We will be exploring how we can leverage NoSQL (MongoDB) to make data management simple and intuitive as well as how it compares to traditional RDBMS systems.

Review: How the Web Works

As we discovered in WEB222, the Internet is the global system of interconnected computer networks that use the Internet protocol suite (TCP/IP) to connect network-enabled devices around the world. We discussed the client/server model and how the client (web browser) makes structured requests for data on the server, which then responds with data (HTML, CSS, Javascript, etc.). This is known as the request/response model and (at it’s most basic level) it consists of a web browser asking the web server to send it a web page and the server sending back the page. The browser then takes care of parsing the HTML and displaying the page:


Web browser
The Internet
Web server at server.com
Disk drive at server.com
1
User enters: http://server.com
2
Look up IP address of server.com
3
Request server.com main page using IP
4
Receive request for index page
5
Fetch index.html from hard disk
6
Return the index page
7
Receive and display page

(https://www.safaribooksonline.com/library/view/learning-php-mysql/9781491906910/ch01.html)

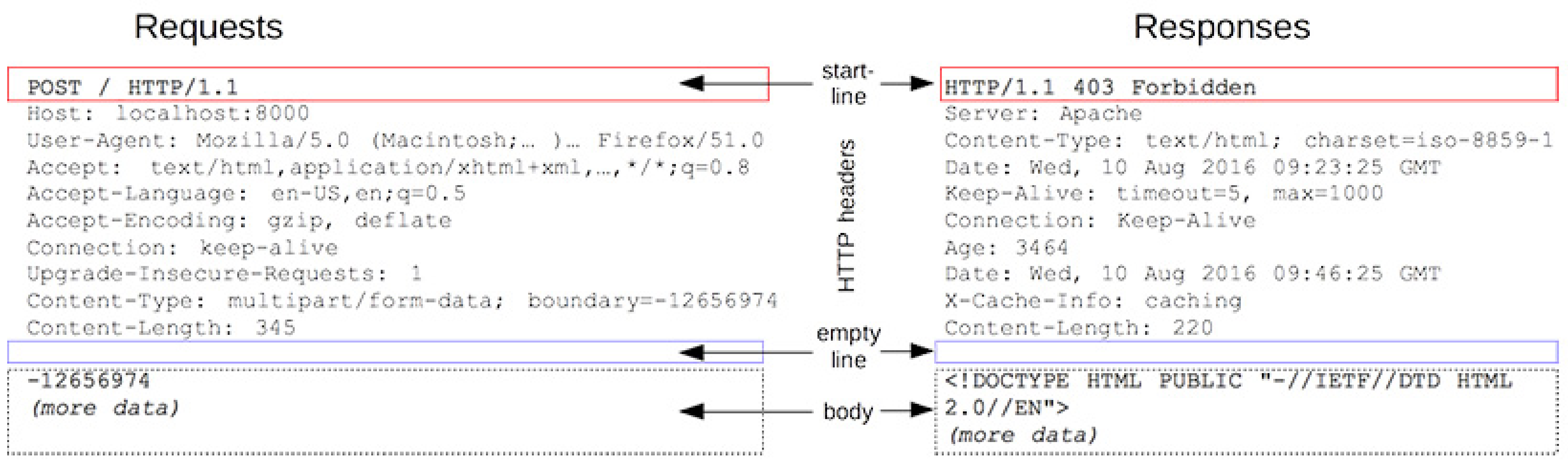
Powering this transaction is a protocol called HTTP (Hypertext Transfer Protocol), which defines a stateless request/response protocol that operates by exchanging messages across a reliable connection. An HTTP “client” (ie: web browser) is a program that establishes a connection to a server for the purpose of sending one or more HTTP requests. An HTTP “server” is a program that accepts connections in order to service HTTP requests by sending HTTP responses.

HTTP Protocol Introduction

The HTTP Protocol itself is an Application layer protocol – that is, it essentially sits “on top” of an underlying network-level protocol such as the Transmission Control Protocol (TCP). HTTP is human-readable and extensible, which makes the protocol extremely easy to extend and to experiment with. New functionality can be introduced simply by establishing an agreement between a client and a server and specifying new “headers” – these will enable the client and server to pass additional information along with the request or the response. The payload content (ie: raw HTML) is sent in the “message body”.

Both HTTP requests and responses share a similar structure and are composed of:

- A start-line that describes the requests to be performed, or its status that is a success or a failure. This start-line is always a single line.
- An optional set of HTTP headers specifying the request, or describing the body included in the message.
- A blank line indicating that all meta-information for the request has been sent.
- An optional body that contains data associated with the request (like the content of an HTML form), or the document associated with a response. The presence of the body and its size is defined by the start-line and the HTTP headers.



```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

```
HTTP/1.1 403 Forbidden
Server: Apache
Content-Type: text/html; charset=iso-8859-1
Date: Wed, 10 Aug 2016 09:23:25 GMT
Keep-Alive: timeout=5, max=1000
Connection: Keep-Alive
Age: 3464
Date: Wed, 10 Aug 2016 09:46:25 GMT
X-Cache-Info: caching
Content-Length: 220

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
2.0//EN">
(more data)
```

(https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)

HTTP Requests

Start line

HTTP requests are messages sent by the client to initiate an action on the server. Their start-line contains of three elements:

1. An HTTP method that describes the action to be performed:

Method	Description
GET	The GET method is used to retrieve information from a specified URI (Universal Resource Identifier) and is assumed to be a safe, repeatable operation by browsers, caches and other HTTP aware components. This means that the operation must have no side effects and GET requests can be re-issued without worrying about the consequences.
POST	The POST method requests that the target resource process the representation enclosed in the request according to the resource's own specific semantics. For example, POST is used for the following functions (among others): * Providing a block of data, such as the fields entered into an HTML form, to a data-handling process * Posting a message to a bulletin board, newsgroup, mailing list, blog, or similar group of articles * Creating a new resource that has yet to be identified by the origin server. * Appending data to a resource's existing representation(s).
PUT	The PUT method is used to request that server store the content included in message body at a location specified by the given URL. For example, this might be a file that will be created or replaced.
HEAD	The HEAD method is identical to GET except that the server MUST NOT send a message body in the response (i.e., the response terminates at the end of the header section). This method can be used for obtaining metadata about the selected representation without transferring the representation data
DELETE	The DELETE method requests that the origin server remove the association between the target resource and its current functionality. In effect, this method is similar to the rm command in UNIX: it expresses a deletion operation on the URI mapping of the origin server.
CONNECT	The CONNECT method requests that the recipient establish a tunnel to the destination origin server identified by the request-target and, if successful, thereafter restrict its behavior to blind forwarding of packets, in both directions, until the tunnel is closed. Tunnels are commonly used to create an end-to-end virtual connection, through one or more proxies, which can then be secured using TLS (Transport Layer Security).
OPTIONS	The OPTIONS method requests information about the communication options available for the target resource. This method allows a client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action.
TRACE	The TRACE method requests a remote, application-level loop-back of the request message. This is typically used to echo the contents of an HTTP Request back to the requester which can be used for debugging purposes during development.

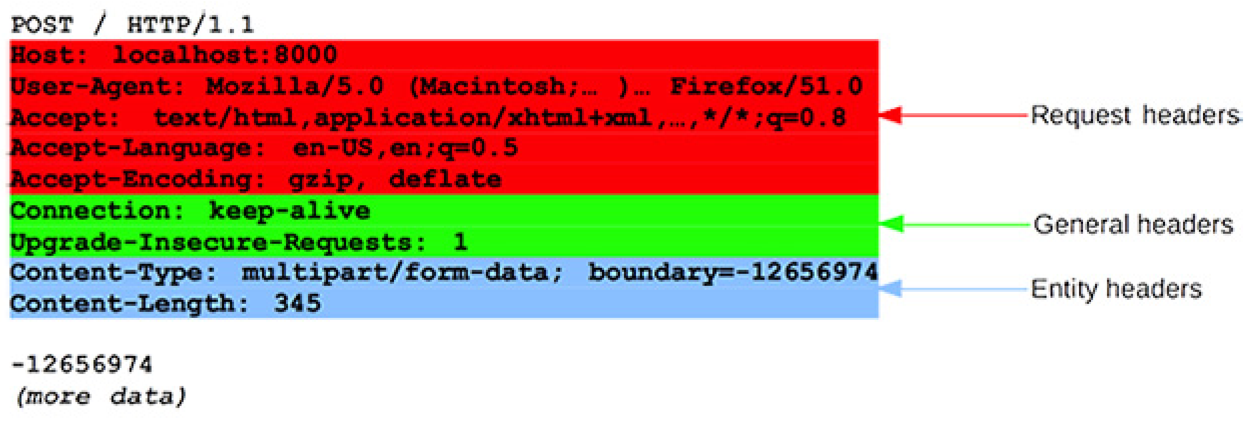
2. The request target (this can vary between the different HTTP methods) – for example, this can be:
    - An absolute path, optionally followed by a '?' and a query string. This is the most common form, called origin form, and is used with GET, POST, HEAD, and OPTIONS methods, for example:
        - **POST / HTTP 1.1**
        - **GET /background.png HTTP/1.0**
        - **HEAD /test.html?query=alibaba HTTP/1.1**
        - **OPTIONS /anypage.html HTTP/1.0**
    - A complete URL, the absolute form, mostly used with GET when connected to a proxy, for example:
        - **GET http://developer.mozilla.org/en-US/docs/Web/HTTP/Messages HTTP/1.1**
    - The authority component of an URL, that is the domain name and optionally the port (prefixed by a ':'), called the authority form. It is only used with CONNECT when setting up an HTTP tunnel, for example:
        - **CONNECT developer.mozilla.org:80 HTTP/1.1**
    - The asterisk form, a simple asterisk ('*') used with OPTIONS and representing the server as a whole, for example:
        - **OPTIONS * HTTP/1.1**
3. The HTTP version, that defines the structure of the rest of the message, and acts as an indicator of the version to use for the response.

Headers

HTTP headers in a request follow the basic structure of any HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consists of one single line, that can be quite long.

There are numerous request headers available. In a request, the headers can be divided in several groups:

- **Request headers:** modify the request by specifying it further, giving context, and/or by conditionally restricting it.
- **General headers:** apply to the message as a whole.
- **Entity headers:** apply to the body of the request (Note: there is no such header transmitted when there is no body in the request).



```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

(https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)

Body

The last part of a request is its body. Not all requests have one: for example, requests fetching resources (like GET or HEAD) usually don't need any. Similarly, DELETE or OPTIONS also do not require a body.

Other requests send data in the body to the server in order to update it: this is often the case of POST requests (that can have HTML form data).

HTTP Responses

Status line

The start line of an HTTP response, called the status line, contains the following information:

1. The protocol version, usually **HTTP/1.1**.
2. A status code beginning with 1, 2, 3, 4 or 5 that provides information such as the success or failure of the request:

Range	Description
1xx	**Informational:** Request received, continuing process. For example, Microsoft IIS (Internet Information Services) initially replies with 100 (Continue) when it receives a POST request and then with 200 (OK) once it has been processed.
2xx	**Success:** The action was successfully received, understood, and accepted. For example, the 200 (Ok) status code indicates that the request has succeeded. The meaning of "success" varies depending on the HTTP method: ◦ **GET:** The resource has been fetched and is transmitted in the message body. ◦ **HEAD:** The entity headers are in the message body. ◦ **POST:** The resource describing the result of the action is transmitted in the message body. ◦ **TRACE:** The message body contains the request message as received by the server
3xx	**Redirection:** Further action must be taken in order to complete the request. For example, The 302 (Found) status code indicates that the requested resource has been temporarily moved and the browser should issue a request to the URL supplied in the Location response header.
4xx	**Client Error:** The request contains bad syntax or cannot be fulfilled. For example, the famous 404 (Not Found) status code indicates that the server can not find requested resource, or is not willing to disclose that one exists.
5xx	**Server Error:** The server failed to fulfill an apparently valid request. For example, the 500 (Internal Server Error) status code indicates that the server encountered an unexpected error / condition that prevented it from fulfilling the request./td>

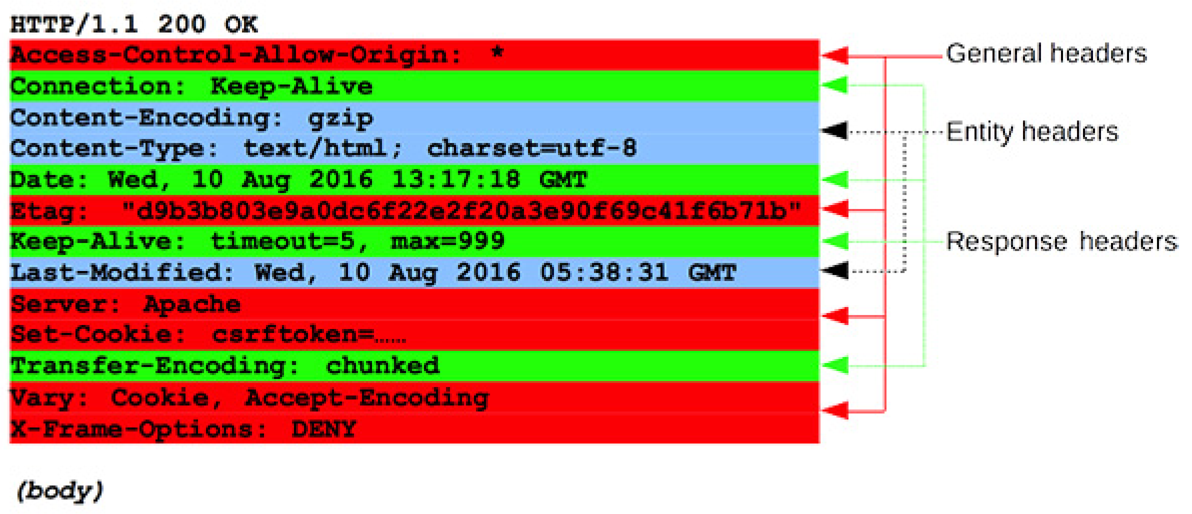
3. A status text, purely informational, that is a textual short description of the status code. This helps HTTP messages be more human-readable, for example:
   - **HTTP/1.1 404 Not Found**

Headers

The HTTP header format for responses follow the same basic structure (a case-insensitive string followed by a colon (':') and a value whose structure depends upon the type of the header. The whole header, including the value, stands in one single line)

There are numerous response headers available. In a response, the headers can be divided in several groups:

- **General headers:** apply to the message as a whole.
- **Entity headers:** apply to the body of the response (Note: there is no such header transmitted when there is no body in the response).
- **Response headers**: give additional information about the server that don't fit in the status line.



```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 10 Aug 2016 13:17:18 GMT
Etag: "d9b3b803e9a0dc6f22e2f20a3e90f69c41f6b71b"
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT
Server: Apache
Set-Cookie: csrftoken=......
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY

(body)
```

(https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages)

Body

The last part of a response is the body. This is typically a single file of known length (defined by the two headers: "Content-Type" and "Content-Length") or a single file of unknown length (encoded in chunks with the "Transfer-Encoding" header set to "chunked". However, not all responses have a body, for example: responses with status code like 201 (Created) or 204 (No Content).

Server Side Programming

The web server plays a key role in not only servicing requests for static content like images and HTML, but also by managing the entire interaction between clients and the server, including things like:

- Authenticating Users / Managing Security
- Redirecting Users
- Handling Errors
- Fetching Data (Flat Files / Database)
- Updating Data
- Handling File Uploads
- Managing real-time chat
- etc…

The development of software to handle all of this is often referred to as "server-side" programming because the "web-server" software resides on the server, rather than the client. In this course we will be creating our own web servers and experimenting with different tools & frameworks to support our development efforts. As mentioned earlier, Node.js will be the runtime enviroment that allows us to execute our web-server code and process HTTP requests. What is interesting about Node.js is that it allows to work exclusively in JavaScript on both the client and the server. We can effectively create web applications by writing JavaScript code on the client that seamlessly connects to our JavaScript code on the server.

Next class we discuss Node.js in greater detail and start writing our first server using JavaScript!

Sources

- http://code.visualstudio.com
- http://blog.chromium.org/
- https://developers.google.com
- https://developer.mozilla.org
- http://es6-features.org
- https://nodejs.org
- https://jquery.com
- https://www.postgresql.org
- https://www.siteground.com
- https://www.safaribooksonline.com/library/view/learning-php-mysql/9781491906910/ch01.html
- https://tools.ietf.org/html/rfc7230
- https://www.w3.org
- https://www.httpwatch.com

- http://www.tutorialspoint.com
