Web Fundamentals - Module 2

This is our final module before we start on the coding and JavaScript topics.

We'll come back to a more in depth look at how web servers work and in particular how HTTP requests and responses are made up.

We'll also cover how to set up a local hosting environment.

This module is quite technical and you instructor will spend a fair amount of time demonstrating and working with you to get your computer configured.

Confidential

Review of Previous Module

In the previous module we looked at some additional CSS selectors, such as the first-child pseudo class and also at attribute selectors.

We discussed inline and block-level elements, the CSS box model, borders and outlines and padding and margins.

We also covered specificity and media types.

Confidential

Topics for this module

In this module we'll look at the following topics:

- Web hosting
- Domain names, IPO addresses and DNS
- The request/response model
- TCP/IP and HTTP
- Virtual hosting
- Configuring a local hosting environment

Confidential

Web Hosting

In the first module we talked about the fact that documents and files (typically referred to as **resources**) that are used for a web site are stored on a computer known as a **web server**.

Web servers need to be permanently available via the **Internet** so that Web clients can connect to them to request Web pages.

Web servers use special programs that listen for incoming requests and then locate and send back the appropriate documents and files for a web page.

There are a wide range of organizations that offer facilities to host websites. These are generally referred to as **Hosting Providers.**

Web Hosting

Before we talk more about hosting providers and the various types of hosting that they offer, lets have a quick recap of the basic processes that are involved in delivering a web page.

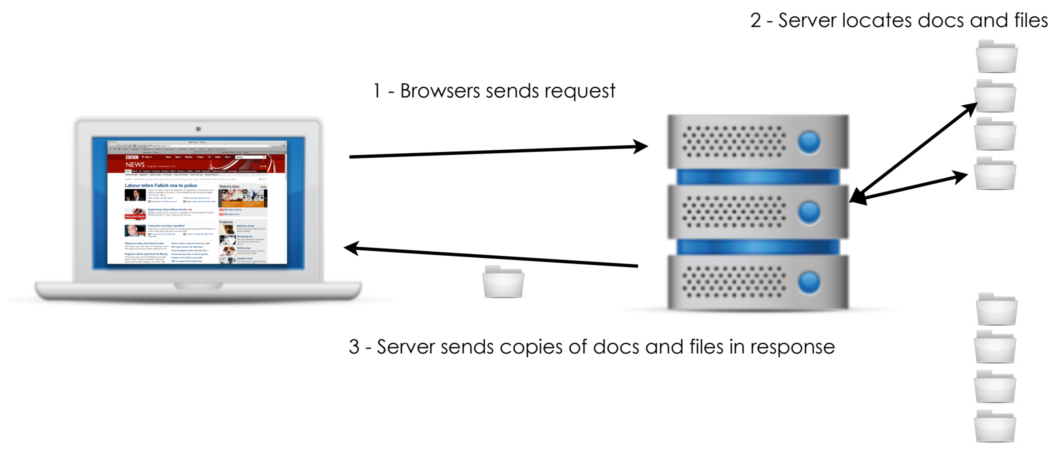
Web Hosting

A browser sends a **request** for a web page to a **Web Server** and then waits for the HTML, CSS and JavaScript documents and file to be sent back as a **response.**

Once the browser receives the various documents and files that make up a page it reads through the HTML markup, CSS styles and JavaScript code and uses them to generate a web page.

This is illustrated on the next slide.

Web Hosting


2 - Server locates docs and files
1 - Browsers sends request
3 - Server sends copies of docs and files in response

Web Hosting

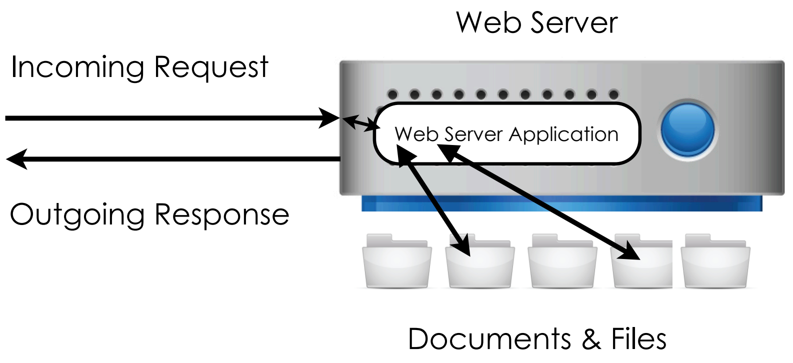
A **Web Server** is a computer that is used to store the documents and files that make up a Web Site.

Web servers need to be permanently available via the **Internet** so that Web clients can connect to them to request Web pages.

Web servers use special programs that listen for incoming requests and then locate and send back the appropriate documents and files for a web page.

The most commonly used Web Server program of this type is called Apache. Others include IIS and Nginx

Web Hosting



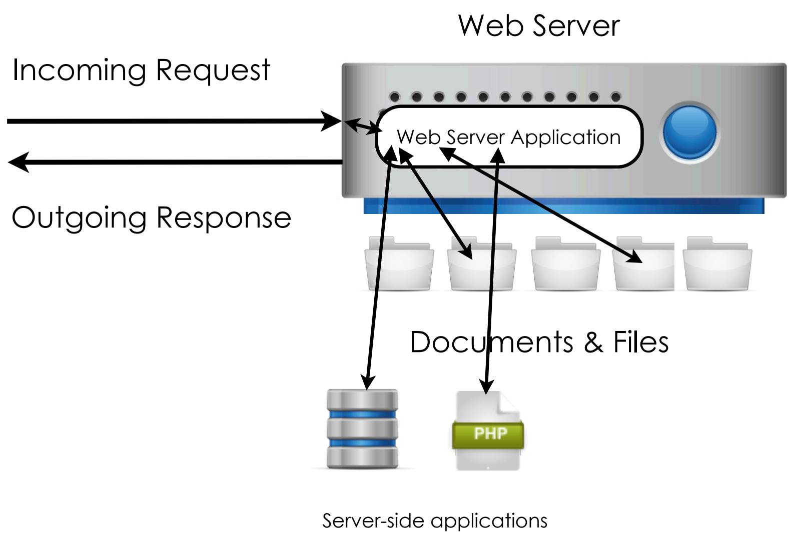
Web Hosting

In addition to storing the files and documents for a Web site, a **Web Server** may have other applications installed that are used to store and generate **dynamic content.**

Such applications might include:

- **Databases** such as MySQL, MongoDb and Oracle
- **Server-side scripting languages** such as Python, PHP & Ruby
- **Development frameworks** such as Django, CodeIgniter & Rails

Web Hosting



Web Hosting

Hosting providers offer a range of options for hosting sites. The most common are:

- Shared hosting
- Virtual Private Server hosting
- Dedicated hosting
- Cloud hosting

Domains, IP Addresses & DNS

IN the first module we said that when you click on a link in a Web page an **HTTP request** is sent to the **Web Server** on which that page is **hosted.**

The Web Server then locates the relevant documents and files for the page and sends them back to the Web client (browser) in the form of an **HTTP Response.**

The link that is clicked, or the address that you enter into the browser, will typically look something like the following:

http://www.bbc.co.uk/news/world/

Let's look at this in a little more detail.

Domains, IP Addresses & DNS

The full address is known as a URL (Uniform Resource Locator)

http://www.bbc.co.uk/news/world/

- The first part '**http**' identifies the **protocol** to be used to send the request – in this case http. Other protocols might be **https** or **ftp.**
- The **protocol** is followed by a '**:**' character
- The two forward slashes '//' indicate that this resource is located on a web server, rather than being on a local drive.
- This is followed by the hostname – **'www.bbc.co.uk'**
- Finally there is a path – '/**news**/**world**/' which identifies a location or file on the web server

Domains, IP Addresses & DNS

Give the example **URL** below

http://**www.bbc.co.uk**/news/world/

- The hostname is 'www.bbc.co.uk'
- **www** is the **subdomain**
- **bbc.co.uk** is the **domain name**
- **uk** is the **top-level domain** (**TLD**)
- **co.uk** is the **second-level domain** (**SLD**)

You can find a detailed breakdown of URLs here:

http://www.mattcutts.com/blog/seo-glossary-url-definitions/

Domains, IP Addresses & DNS

We've already seen that a single web server can be used to host several websites. This could cause a problem if the domain name of a particular site were used to identify a web server as it would limit that server to hosting a single site.

Instead each web server is identified by an IP Address

Multiple domains can then be assigned to a common IP address, thereby allowing sites for those domains to be located on the same server.

In order for this to work there needs to be a way of storing the IP address for any given domain, so that the HTTP request can be direct to the correct web server.

There also needs to be a way that the web server can locate the resources for any given domain that it hosts.

Domains, IP Addresses & DNS

IP addresses are represented in dot-decimal notation.

This consists of four decimal numbers, each ranging from 0 to 255, separated by dots.

For example:

146.12.254.1

You can think of an IP address as being like a phone number for a computer. Once we know the IP address we can make a call (send a HTTP request) to the web server that it identifies.

All we need now is some kind of directory enquiry service so that we can find the IP address for a given domain.

Domains, IP Addresses & DNS

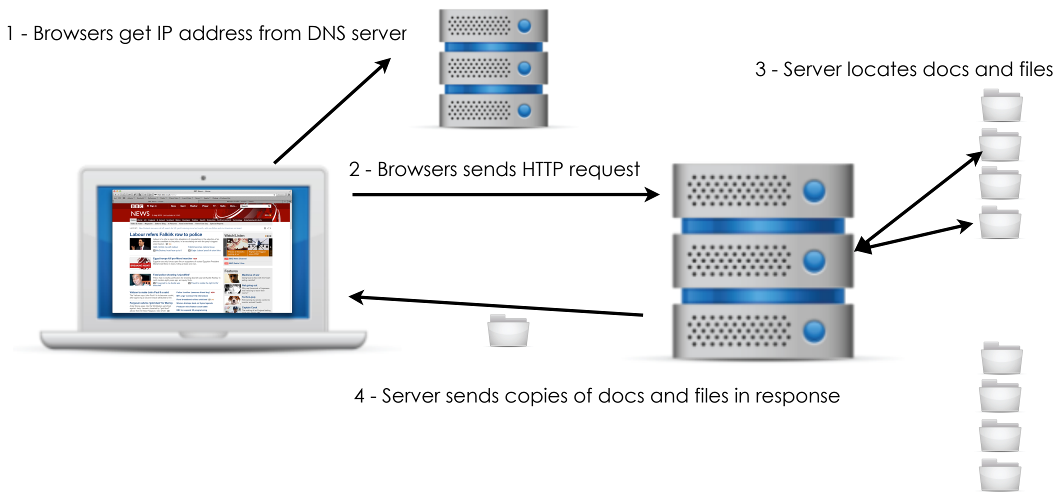
The **Domain Name System**, or **DNS**, is a system (protocol) that is used to relate a **domain name**, such as **bbc.co.uk**, to an **IP address** like **212.58.253.67**

The Domain Name System uses special servers, known as **DNS servers** or **name servers**, which store listings of domains and their associated IP addresses – in much the same way that a telephone directory lists people's names and phone numbers.

When a web client such as a browser needs to send a request for a particular web site it submits the domain to a **DNS server** which then sends back the **IP address**. This is known as **DNS name resolution**.

IP addresses can be specified for the domain itself and also for any **subdomains**, so you think of a subdomain as being a bit like an extension for a main phone number.

Domains, IP Addresses & DNS


1 - Browsers get IP address from DNS server
3 - Server locates docs and files
2 - Browsers sends HTTP request
4 - Server sends copies of docs and files in response

Virtual Hosting

We've seen that the Domain Name System allows a domain name to be resolved into an IP address, and the IP address can then be used to identify the web server on a which a site is hosted.

The final piece of the jigsaw is that a single web server will be receiving requests for many different sites and pages – so it needs a way of knowing where the files for any given domain or subdomain are located.

This is usually done by a process know as virtual hosting, which allows a single web server to act as a hosts for multiple sites.

We'll look at how this works on a server that is running the Apache web server.

Virtual Hosting

Apache has a number of configuration files that it uses to manage the various processes.

Of these the most important from a web development and hosting perspective are:

- The **main configuration file** – usually named ‘**httpd.conf**’
- The **virtual hosts configuration file** – usually named ‘**httpd-vhosts.conf**’ or similar

We’ll look at Apache’s main configuration file later and for now we’ll focus on the virtual hosts configuration file.

Virtual Hosting

A virtual hosts configuration file is made up of a number of sections.

Each section contains directives that Apache uses.

The sections are identified by opening and closing ‘tags’

When Apache receives a request for a particular server name it will use that to lookup the related directives and settings.

Apache reads this configuration file when it is started and then stores all the directives and setting in memory.

If the configuration file is modified then it is necessary to restart Apache so that it reads and stores the new version.

Example from a virtual hosts file

```
NameVirtualHost *:80
#
<VirtualHost *:80>
    DocumentRoot /var/www/vhosts/example1/httpdocs
    ServerName www.example1.com
</VirtualHost>
```

Location of files (root directory)

Server name = subdomain +domain

```
<VirtualHost *:80>
    DocumentRoot /var/www/vhosts/example2/httpdocs
    ServerName www.example2.com
</VirtualHost>
```

Location of files (root directory)

Server name = subdomain +domain

Your instructor will talk through this with you and show a more detailed example.

TCP/IP and HTTP

In the first module we said that **HTTP** is the **protocol** that is used for sending requests and responses for web pages.

In fact, HTTP is one of a family of communication protocols listed below:

- Internet Protocol (IP)
- Transmission Control Protocol (TCP)
- Hypertext Transfer Protocol (HTTP)

This family of protocols is known as the **TCP/IP protocol**, or **TCP/IP** for short.

The Request / Response Model

The HTTP protocol is used for sending requests from a web client to a web server and the associated response from the web server back to the client.

There are currently two versions of the HTTP protocol:

- **HTTP/1.0** – this is the original version of the protocol
- **HTTP/1.1** – a later version of the protocol that incorporates additions functionality

When a web client sends an HTTP request, or a web servers sends a HTTP response, it must identify the version that it is using.

The most commonly used protocol is HTTP/1.1

HTTP Requests

You can think of an HTTP request as being similar in concept to letters that you send through the mail to request some information.

As a minimum in needs to have an address that the request is being sent to and some details of what is being requested.

A correctly structured HPPT request contains the following components:

- A request line.
- A number of HTTP headers.
- An optional message body.

Each **HTTP header** is followed by a carriage return line feed (CRLF). An additional CRLF is used after the last HTTP header, to create an empty line, and then the **message body** starts if there is one.

HTTP Request – Example

An example HTTP request is shown below. In this example there is no data being sent.

```
GET /news/index.html HTTP/1.1
Host: www.example1.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4)
Referer: http://www.wombats.com/
[blank line here]
```

This header is using the **GET method** of version **1.1** of the HTTP protocol to send a request to the **web server** that is **hosting** the **www.bbc.co.uk**

In addition it is specifying a **path** of **/news/index.html** to locate the appropriate resource. Additional **headers** provide information about the **type of browser** that is being used (**User-Agent**) and the site from which the **request originated** (**Referer**)

HTTP Request – Example

The receiving web server application (Apache) will then read the headers to identify and locate the resources that are being requested.

```
GET /news/index.html HTTP/1.1
Host: www.example1.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4)
Referer: http://www.wombats.com/
[blank line here]
```

2. Next, the path from the request is appended to the DocumentRoot to locate the resource

```
<VirtualHost *:80>
    DocumentRoot /var/www/vhosts/example1/httpdocs
    ServerName www.example1.com
</VirtualHost>
```

1. First, Apache uses the host header to lookup the ServerName

HTTP Responses

An HTTP response is also similar in concept to a package that is sent through the mail.

A correctly structured HPPT response contains the following components:

- A status line.
- A number of HTTP headers.
- An message body.

Each HTTP header is followed by a carriage return line feed (CRLF). An additional CRLF is used after the last HTTP header, to create an empty line, and then the message body starts.

Note: HTTP responses will almost always have a body component.

HTTP Response – Example

An example HTTP response is shown below.

```
HTTP/1.1 200 OK
Date: Mon, 08 Jul 2013 10:14:06 GMT
Server: Apache
Content-Type: text/html
[blank line here]
<DOCTYPE html>
<html>
  (remaining HTML here...)
</html>
```

This response is using HTTP version **1.1** and is sending a **status code** of 200 and a **reason phrase** of OK.

In addition it is identifying the **date and time** that the response was sent, the **type of web server** that sent it and the **type of content**.

Finally it sends a blank line and then the body – in this case HTML

Local Hosting

So far, everything that we have discussed in this module makes the assumption that a remote web server is being used to host the files and resources for a site, and this will certainly be the case for a live public-facing site as it needs to be on a server that is permanently connected to the internet.

However, when developing web sites and applications it can be useful to use your own computer to act as a web server – so that you can develop and test in a private and safe environment.

In the next few slides we'll look at how that can be achieved, and your instructor will demonstrate and provide additional details.

Local Hosting – Apache

First of all we need a web server application to be available and running on the local computer. There are a number of web servers that we could use but for this course we'll focus on **Apache**.

If you are using an Apple Mac, MacBook or any Linux/Unix based computer then the good news is that Apache is already installed and we just need to fire it up.

If you are using a Windows based computer then it is likely that Apache is not installed.

EIther way you will be provided with instructions as to how to install Apache if necessary and then how to start and test that it is working.

Your instructor will walk through this now to make sure that everyone has Apache ready to use.

Local Hosting – Apache – It Works!

Once you have Apache installed and running you should be able to enter the following URL in a browser and see a default Apache page – often with a message saying 'It works!'

`http://localhost`

We're (almost) ready to start hosting sites locally, but there are a couple more things that we need to cover first:

- Apache's **DocumentRoot** setting
- Use of a local '**hosts**' file

Let's take a look at these now...

Apache – Configuration file

In an earlier slide we looked at some example entries in Apache's virtual hosts configuration file. We'll now look at an important entry in Apache's main configuration file.

On most Mac/Linux/Unix systems the main configuration file for Apache is named '**httpd.conf**' and it is located in a directory (folder) named: /etc/apache2

The path to the file would therefore be:

```
/etc/apache2/httpd.conf
```

On Windows the equivalent path will most probably be one of the following – depending on which version of Apache is installed:

```
C:\Program Files\Apache Group\Apache2\conf\httpd.conf
C:\Program Files\Apache Software Foundation\Apache2.2
```

Apache Configuration file

The Apache configuration file is **critical** to the correct operation of Apache.

For this reason it is **strongly recommended that you never edit this file** unless you absolutely have to, and that if you do you should always make a safe copy that you can revert to if anything goes wrong.

There are other ways to modify directives without editing the main file, which we'll cover on the Bootcamps.

Just to be absolutely clear:

Do not edit Apache's configuration file unless you absolutely have to AND you know what you are doing. If you do edit it then ALWAYS take a copy first.

Apache – Configuration file

Having given warnings about the potential dangers of editing Apache's main configuration file you will likely find that it is set with file permissions that have to be changed before you could edit it anyway.

As far as the process of setting up a local server is concerned all we want to do is to look at one of the entries in the file.

The entry that we're interested is the **DocumentRoot** setting. An example is shown below:

```
DocumentRoot "/Library/WebServer/Documents"
```

Your instructor will show you how to locate this setting.

Apache – DocumentRoot

The Document Root, which is set with the DocumentRoot directive in Apache's main configuration file, tells Apache where to look for web site documents and resources by default.

```
DocumentRoot "/Library/WebServer/Documents"
```

(Note: this can be overwritten for any particular ServerName in a virtual hosts configuration file as we saw earlier)

Make a note of this location. If you look in that folder you will most likely find some example files already there.

Apache – localhost

To test that Apache was working we entered the the following URL in a browser:

```
http://localhost
```

And, magically, a page was displayed. The page was **served** by Apache from the default **DocumentRoot** location.

However, why didn't the browser do what it normally does, and use the Domain Name System (DNS) to find the IP address for a server named 'localhost'

Any ideas?

The ‘hosts’ file

We said before that when a URL is entered in the address bar of a browser, or a link is clicked on a web page, that the browser will use the Domain Name System to find the IP address for the server.

In actual fact it it does something just before it tries to use DNS.

What it does is to look for a file named ‘**hosts**’ on the **local computer** (there is no file extension)

If this file exists (which it usually will) then the browser will look in there to see if it can find an IP address for the server that the request is for.

If it can then it will use that IP address and not bother with DNS.

The ‘hosts’ file – Example

Here is an example of a default hosts file.

Your instructor will talk through this with you.

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1        localhost
255.255.255.255  broadcasthost
::1              localhost
fe80::1%lo0      localhost
```

The ‘hosts’ file – location

The hosts file is usually located as follows:

On Macs:

```
/etc/hosts
```

On Windows:

```
c:\windows\system32\drivers\etc\hosts
```

The ‘hosts’ file – format

The hosts file comprises a number of entries. Each entry is located on a new line and takes the format:

```
IP Address    servername
```

It also includes some special entries at the top:

```
127.0.0.1           localhost
255.255.255.25      broadcasthost
::1                 localhost
fe80::1%lo0         localhost
```

The first of these associates the IP address 127.0.0.1 with the name ‘localhost’. 127.0.0.1 is the IP address for your local computer – so the browser send the HTTP request to the local instance of Apache rather than looking it up via DNS.

The ‘hosts’ file – adding entries

The hosts file is also an important file but is is common practice to edit it as part of the process of setting up a local hosting environment.

It is always good practice to make a copy of the file first – just in case anything goes wrong.

Your instructor will now demonstrate making a new entry in the hosts file.

The ‘hosts’ file – adding entries

```
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1         localhost
255.255.255.255   broadcasthost
::1               localhost
fe80::1%lo0       localhost

# Add a new entry for a local site
127.0.0.1        local.mytestsite.com
```

Comment

Entry

The ‘hosts’ file – adding entries

Now that we’ve added the entry in the hosts file, let’s see what happens when we use our new server name in a UTRL.

Enter the following in your browser:

```
http://local.mynewsite.com
```

You should see the same default ‘It works!‘ page as before – which isn’t quite what we want.

The browser is seeing the new entry in the hosts file but there is nothing to tell Apache where the files for that site are located, so it it is using the default DocumentRoot setting.

Let’s change that...

Add a virtual hosts entry

We said that the virtual hosts configuration file contained setting that told Apache to find the files for a given server.

If Apache can’t find a section for the server it will use the first section in the file, or the default in the main configuration file.

Lets add an entry for our new server:

```
<VirtualHost *:80>
    DocumentRoot /path/to/your/files
    ServerName local.mynewsite.com
</VirtualHost>
```

The location on your computer

We’re nearly done...

Add a virtual hosts entry

Finally, create a new HTML file named ‘index.html‘ in the directory that you have specified.

Add the following markup code to the body element:

```
<h1>Welcome to my test site</p>
<p>This is my new site - it’s hosted on a local
sever...</p>
```

Then restart Apache and try the URL again.

Module Summary

In this module we looked at some additional CSS selectors, such as the first-child pseudo class and also at attribute selectors.

We discussed inline and block-level elements, the CSS box model, borders and outlines and padding and margins.

We also covered specificity and media types.

Confidential

Labs

Your instructor will upload the labs for this module to Basecamp after the session ends.

Confidential

Questions

Do you have any questions before we move on to the next module

?

Don’t worry, if you think of something later then just ask!