

ICTWEB503

Create web-based programs

Learner Guide





© Copyright, 2015 by North Coast TAFEnow

Date last saved: 16 October 2015 by Amanda Walker	Version: 1	# of Pages = 57
Greg Bell – Content writer and course adviser TAFEnow Resource Development Team – Instructional and graphic design		

Copyright of this material is reserved to the Crown in the right of the State of New South Wales.

Reproduction or transmittal in whole, or in part, other than in accordance with the provisions of the Copyright Act, is prohibited without written authority of North Coast TAFEnow.

Disclaimer: In compiling the information contained within, and accessed through, this document ("Information") DET has used its best endeavours to ensure that the Information is correct and current at the time of publication but takes no responsibility for any error, omission or defect therein. To the extent permitted by law, DET and its employees, agents and consultants exclude all liability for any loss or damage (including indirect, special or consequential loss or damage) arising from the use of, or reliance on, the Information whether or not caused by any negligent act or omission. If any law prohibits the exclusion of such liability, DET limits its liability to the extent permitted by law, to the re-supply of the Information.

Third party sites/links disclaimer: This document may contain website contains links to third party sites. DET is not responsible for the condition or the content of those sites as they are not under DET's control. The link(s) are provided solely for your convenience and do not indicate, expressly or impliedly, any endorsement of the site(s) or the products or services provided there. You access those sites and use their products and services solely at your own risk.

Contents

Getting Started	i
About this unit	i
Elements and performance criteria.....	i
Icon Legends.....	ii
Topic 1 – The development environment	1
“AMP”	1
Other operating systems.....	3
Running WAMP.....	3
Finding your way around WAMP	4
Testing.....	5
Topic 2 – Understanding sessions	7
HTTP	9
Hacking HTTP	11
Installing a terminal client	11
Watching HTTP work	15
Tools to examine open connections	17
Browser-based developer tools.....	19
The Stateless web	21
Topic 3 – The Stateful web	28
URLs as storage	28
Cookies as storage	32
Our more sophisticated example	34
Sessions and cookies	37
Other browser storage schemes	43
Terminating sessions	44
Topic 4 – Security threats	46
Sensitive information	46



User tinkering.....	47
Tinkering in transit.....	47
Encrypting.....	47
Signing.....	47
Hashing.....	47
Verifying IP.....	48
Spoofing / replay.....	48
Cookies-only.....	48

Getting Started

About this unit

This unit describes the skills and knowledge required to develop Web Applications.


It applies to individuals who work as web developers and have well-honed technical skills to take responsibility for implementing code required to create Web Applications.

Elements and performance criteria

Elements define the essential outcomes of a unit of competency. The Performance Criteria specify the level of performance required to demonstrate achievement of the Element. They are also called Essential Outcomes.

Follow this link to find the essential outcomes needed to demonstrate competency in this Unit: <https://training.gov.au/Training/Details/ICTWEB503>

Icon Legends

	<p>Learning Activities</p> <p>Learning activities are the tasks and exercises that assist you in gaining a clear understanding of the content in this workbook. It is important for you to undertake these activities, as they will enhance your learning.</p> <p>Activities can be used to prepare you for assessments. Refer to the assessments before you commence so that you are aware which activities will assist you in completing your assessments.</p>
	<p>Case Studies</p> <p>Case studies help you to develop advanced analytical and problem-solving skills; they allow you to explore possible options and/or solutions to complex issues and situations and to subsequently apply this knowledge and these newly acquired skills to your workplace and life.</p>
	<p>Discussions/Live chat</p> <p>Whether you discuss your learning in an online forum or in a face-to-face environment discussions allow you to create and consolidate new meaningful knowledge.</p>
	<p>Readings (Required and suggested)</p> <p>The required reading is referred to throughout this Learner Guide. You will need the required text for readings and activities.</p> <p>The suggested reading is quoted in the Learner Guide, however you do not need a copy of this text to complete the learning. The suggested reading provides supplementary information that may assist you in completing the unit.</p>
	<p>Reference</p> <p>A reference will refer you to a piece of information that will assist you with understanding the information in the Learner Guide or required text. References may be in the required text, another textbook on the internet.</p>
	<p>Self-check</p> <p>A self-check is an activity that allows you to assess your own learning progress. It is an opportunity to determine the levels of your learning and to identify areas for improvement.</p>
	<p>Work Flow</p> <p>Shows a logical series of processes for completing tasks.</p>



Topic 1 – The development environment

“AMP”

Luckily for us, Apache, MySQL, and PHP are so popular that they have been given their own name - “AMP” - and they are packaged together in special distributions. Even better, one of these distributions is called “WAMP”, with the “W” standing for Windows. A tremendous amount of work has been done for us already, available at the click of a mouse.

**Install WAMP**

To install WAMP, go to this URL and download it.

<http://www.wampServer.com/en/>

Should this link be unavailable please report to TAFENow and instead search the internet for "WAMP Server download"

Make sure to pick the appropriate version (32 or 64 bit) for your operating system. Installing the 64 bit version in a 32 bit operating system will result in runtime errors.

At the time of this writing Visual C++ 2010 SP1 Redistributable Package was required by WAMP. This can be downloaded from Microsoft.

VC10 SP1 vcredist_x86.exe 32 bits:

<http://www.microsoft.com/download/en/details.aspx?id=8328>

VC10 SP1 vcredist_x64.exe 64 bits:

<http://www.microsoft.com/download/en/details.aspx?id=13523>

Should these links be unavailable please report the issue to TAFENow and instead search the internet for "Microsoft Visual C++ 2010 Redistributable Package"

Many online postings indicated that the 2008 version of the C++ redistributable was simultaneously required:

<http://www.microsoft.com/en-us/download/details.aspx?id=29>

There are several other considerations. The Web Server will occupy TCP/IP port 80, which means you should not have any other applications talking or listening on that port. If you have Windows' Internet Services installed and running, they will also be trying to use port 80, which will cause a problem. Disable it. Skype often uses port 80. Other applications that use the Internet might as well. Disable or exit any suspects.

Because this is an entire Web Server we are installing, there is quite a bit it needs to do to your system. The installer wants to make changes to your registry and to your C:\Windows\System32/etc/drivers/hosts file. So, you might find that your antivirus program needs to be disabled during installation.

During installation, Windows Firewall should offer to open Apache's access to your machine / network. Allow it.

At the end of the installation, the installer will offer to launch WAMP immediately. Do this and see if it worked. If you get a runtime error, see this page of troubleshooting tips:

<http://forum.wampServer.com/read.php?2,71076>

Should these links be unavailable please report the issue to TAFENow and instead search the internet for "WAMP Server home forum"

And of course, search the internet for your symptom if the above does not help.

At the time of this writing, the following versions were installed by WAMP:

- > Apache 2.4.4
- > MySQL 5.6.12
- > PHP 5.4.16

Other operating systems

Linux and OSX are an even more natural fit for the AMP stack. For details on installing on those operating systems, search the internet.

Running WAMP

You might want to tell Windows to always show that icon via Control Panel\All Control Panel Items\Notification Area Icons.

It looks like this:

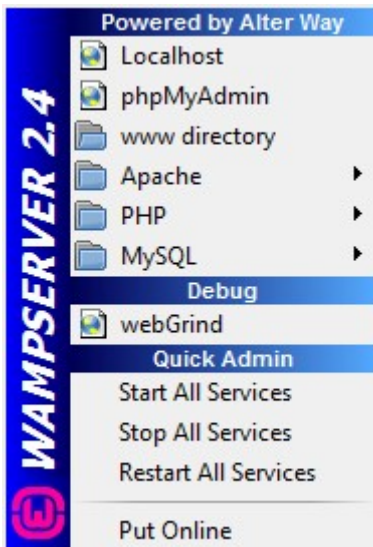


**Launch WAMPServer**

If you didn't launch WAMPServer at the end of the installation, launch it now from the Start Menu. Verify the icon shows up in your system tray.

Finding your way around WAMP


The WAMP menu (left-click on the tray icon) provides a quick way to start and stop the Servers (MySQL and Apache). It also provides lots of shortcuts to important places on your hard disk, for example the directory that it serves HTML and PHP files from, and various log files kept by the running processes.

**WAMP**

Explore all the WAMP menu options.

Testing


No setup is required for our purposes in this unit. Apache's `httpd.conf` and `php.ini` are all ready to go. A debugger called XDebug, which improves the output to your browser when your PHP scripts die, is even included and configured.


LEARNING ACTIVITIES

ACTIVITY 4

Test WAMP

To test WAMP, point your browser to <http://localhost>. You should see this page:


WampServer

Version 2.4 Version Française

Server Configuration

Apache Version : 2.4.4
PHP Version : 5.4.16
MySQL Version : 5.6.12

Loaded Extensions :

Core	bcmath	calendar	ctype	date
ereg	filter	ftp	hash	iconv
json	mcrypt	SPL	odbc	pcre
Reflection	session	standard	mysqlnd	tokenizer
zip	zlib	libxml	dom	PDO
Phar	SimpleXML	wddx	xml	xmlreader
xmlwriter	apache2handler	gd	mbstring	mysql
mysqli	pdo_mysql	pdo_sqlite	mhash	xdebug

That's a Web Server running on your local machine serving you a page from your local disk!

If that wasn't exciting enough for you, note the number of extensions included in WAMP's packaging of Apache. That's an amazing amount of software provided to you for no cost and for very little work by us.

5 | Page
ICTWEB503_LG_V1.1.DOCM
TAFE now

**Create a message**

Navigate to C:\wamp\www and rename `index.php` to `wamp_dashboard.php`. Create a file called `wamp_dashboard.php` and put this in it:

```
<html>
<body>
  Hello from the Server
</body>
</html>
```

Now refresh your Web Browser pointing to `http://localhost` and you should see your message.

Now you will have some appreciation for the open source solutions out there - you have a complete Web Application stack installed locally, for free, with no licensing requirements. Next we'll dive into HTTP and look at some low level ways of observing how it works.



Topic 2 – Understanding sessions

Now that we have a local Web Server, we'll develop a mental model of how the web works at a low level, and see how HTTP works. We will explore the browser's Developer Tools and see how they can be useful to watch low level behaviour of the browser and Web Server. Finally, we'll learn about sessions and states and discover that the base web technologies don't really provide the state storage required for modern Web Applications.

A session is a time devoted to a particular activity. For example, here's a phone call to the bank.

```
<Dial, connect>
You: "Hello?"
Bank: "Hello, this is the bank"
You: "What are your opening hours today?"
Bank: "9am to 5pm"
You: "And where are you located?"
Bank: "195 State St"
<Hang up>
```

(We're modelling a back-and-forth request and response protocol, so that's why the first line looks a little funny.)

What if you have another question tomorrow? You're a bit selfish and you'd prefer the bank just stay on the line so they answer any future requests quickly. This takes resources (a phone line and a staff member).

But it also takes time to dial, say hello, etc. So clearly there is a trade-off to be made. We'll look at this more later when we look at how HTTP works.

Imagine if after every question and response, the phone connection was terminated. And say the conversation was slightly different.

```
<Dial, connect>
You: "Hello?"
Bank: "Hello, this is the bank"
<Hang up>
<Dial, connect>
You: "What are your opening hours today?"
Bank: "9am to 5pm. But our Brentwood branch is open until
6pm."
<Hang up>
<Dial, connect>
You: "Where's that one?"
Bank: "195 State St"
<Hang up>
```

Keep this in mind in the following sections, because that's exactly how the web works. As impractical as it seems that's the way things need to work, as we'll see.

This is a very important distinction – the session duration is different than the connection duration. The session lasts longer than the connection does. On the web, your session may last days, weeks or even years (for a login), but your connection gets terminated after every page load.

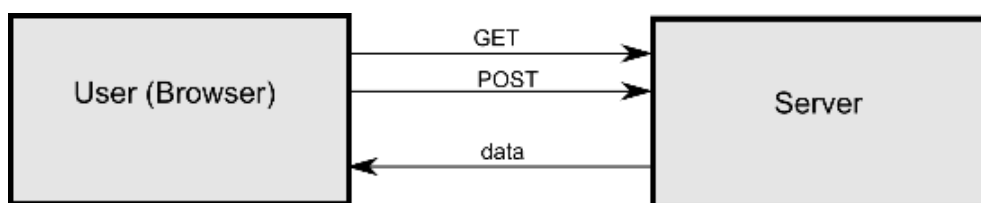
A particular problem is the last question: "Where's that one?" One what? Understanding that question depends on the answer given during the former call with the bank: "But our Brentwood branch is open until 6pm". With that last phone call, how does the bank know what you're talking about? They need memory of the previous conversation they had with you.

HTTP

Let's take a look at how HTTP – the Hypertext Transfer Protocol – works. This will help us understand states and sessions later.

Communication on the web

With communications on the web, the user (human) uses a Web Browser to issue various HTTP commands to the Web Server. These are usually GET or POST commands. The Server responds with data, which can be images, HTML, JavaScript, CSS or other resources, and the browser uses that to render the Web Page.



With the original HTTP specification, the connection was terminated after every request to free up network resources. This was fine when pages were simple, but soon pages were pointing to other resources, which also needed to be fetched with GET requests. Like in this example page:

```
1 <!doctype html>
2 <head>
3   <title>Sample page</title>
4   <link rel="stylesheet" type="text/css" href="css/base.css"
5     media="all">
6 </head>
7 <body>
8   <div class="top-wrappper">
9     <header id="header" class="global-width">
10    .
11    .
12    
14    .
15    .
16    <script src="js/jquery.carouFredSel-5.6.1-packed.js"
17      type="text/javascript" charset="utf-8"></script>
```

That one page contains links to several other resources: some CSS (line 4) an image (line 12), and a JavaScript file (line 16). All of these resources are also fetched using a GET request.

It would be impractical if the Server closed the connection after every GET request, requiring the user's machine and the Server, and all hardware in between, to re-establish the connection for each resource. There would be a lot of connection set-up and tear-down overhead for this one page.

HTTP 1.1 established that the connection should be kept open for certain amount of time after a GET has been processed, allowing additional GETs to be serviced without re-establishing the connection. This lets at least all of the resources for one page be loaded over one connection (like getting all of your questions answered with one phone call to the bank), and maybe even the next page if the user clicks on the next link quickly.

This the trade-off settled on between keeping the resource tied up to keep the connection open, and the overhead involved in establishing a new connection. Connections usually stay open a minute or so.

Let's convert our above example to the web equivalent:

```
<Connect via TCP/IP>
You: "GET me your homepage"
Bank: "here it is."
You: "I see the page includes the bank logo. GET me that
image."
Bank: "Here it is."
You "I see the page includes a CSS file. GET me that."
Bank: "Here it is."
<Some delay>
<Connection terminated>
```

This is a protocol between computers, so we don't even need to say "please" or "thank you".

It's at this point that all the resources needed to display the page have been given to the Web Browser. So it's a good time to terminate the connection while the user decides what to click on next.

Hacking HTTP

Let's see what this looks like at a really low level. We are going to directly communicate with our WAMP Server. To do this, we need a simple program which will allow us to send raw bytes to Apache, and see the response.

Below are several options for doing this, listed in decreasing order of preference. You only need one.

Installing a terminal client

Telnet in OSX or Linux


Telnet is a "terminal program" leftover from the old days when all interaction with computers was text-based. Telnet simply sends characters to, and receives characters from a Server of some sort. We're going to use it to send commands to the Web Server directly, acting like a browser.

This sort of low level interaction has gone out of fashion, but is still useful to see principles at work, and for debugging.

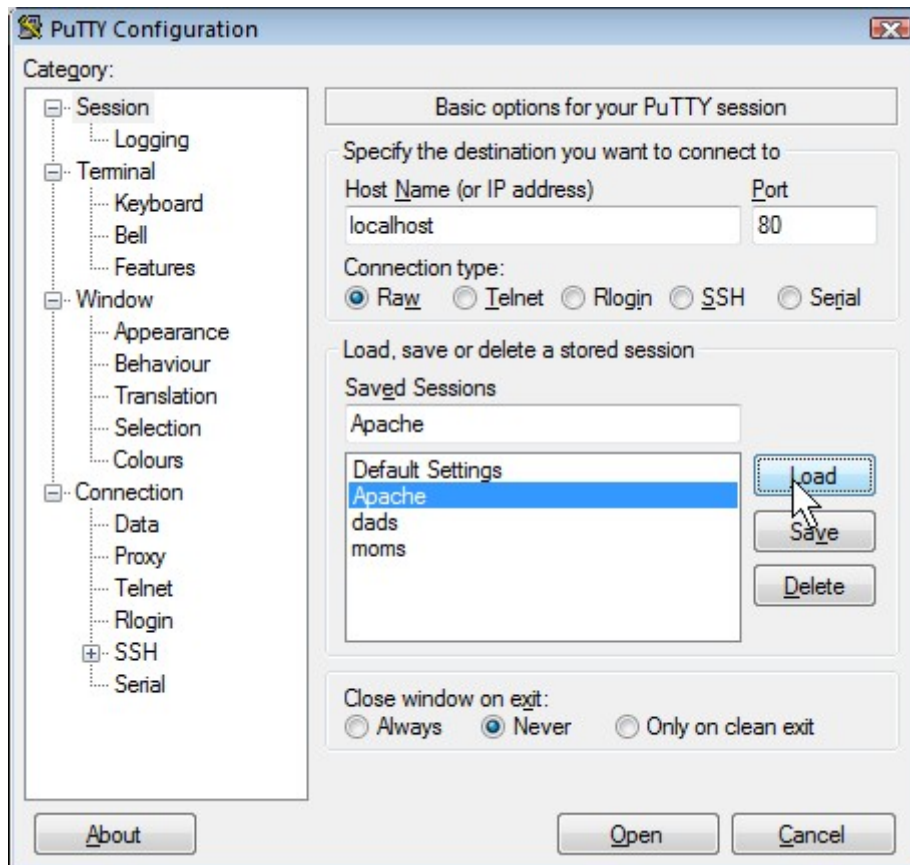
OSX or Linux users can use telnet by just typing 'telnet' in a terminal window. If it is not installed, search your package repositories for it.

Putty in Windows

Putty is a great general-purpose telnet (and secure shell) program.

	LEARNING ACTIVITIES	ACTIVITY 6
<p>Putty</p> <p>Download putty from http://www.chiark.greenend.org.uk/~sgtatham/putty/</p> <p><i>Should this link be unavailable please report the issue to TAFENow and instead search the internet for "PuTTY Free Telnet download"</i></p>		

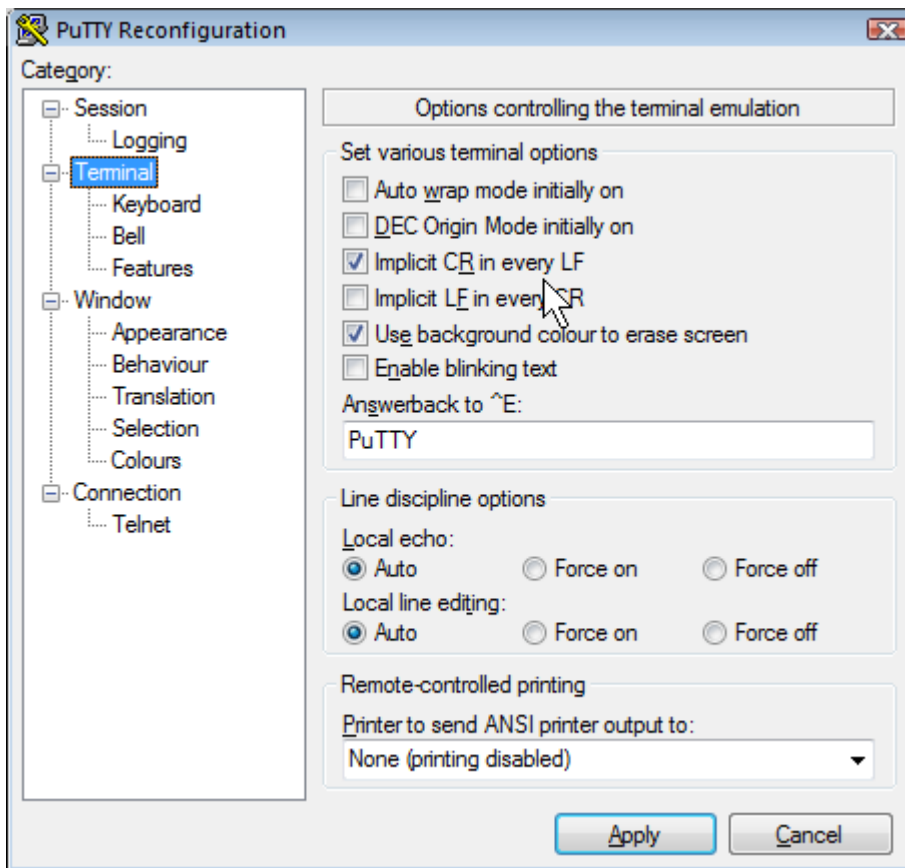
Setting up the connection in Putty would look like this:



The Connection Type is set to “Raw” since we just want sending-and-receiving of bytes. This is a bit confusing since Telnet is actually a protocol, but many programs are called ‘telnet’ that just send and receive raw bytes. Apache doesn’t speak telnet – it speaks HTTP over a raw (protocol-less) communications channel.

Also set the “Never” setting for “Close window on exit”. Since the connection is terminated after a response from the Server, this setting keeps the window open so you can restart the session by clicking on the menu in the Putty Icon in the window’s title bar.

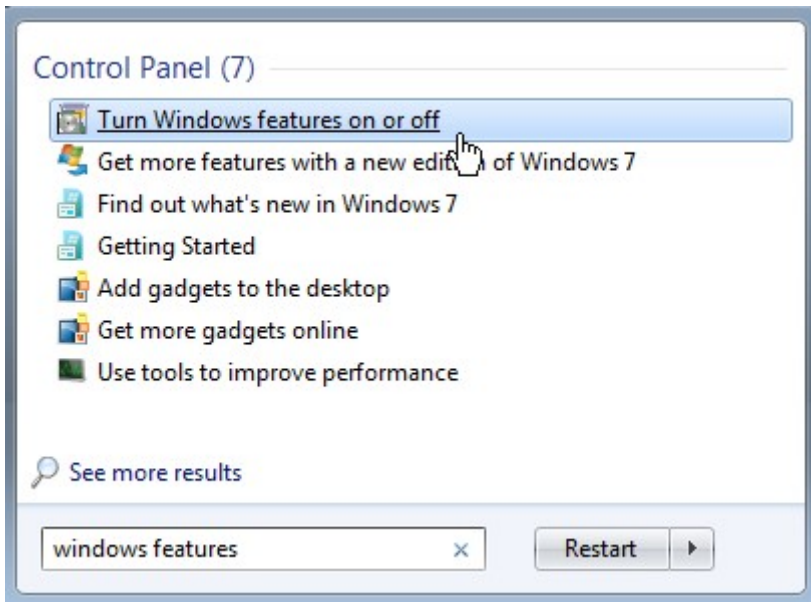
Also set the “Implicit CR in every LF” which will keep the output looking reasonable.



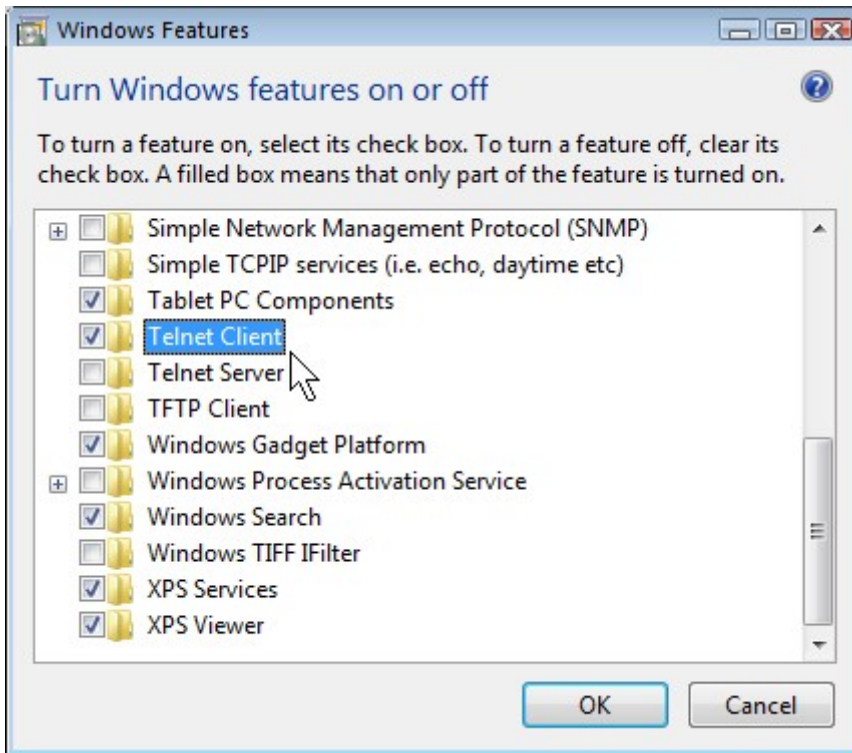
Windows' Telnet

Important note: At the time of this writing, Microsoft Telnet had a disagreeable bug where it would put the cursor at the top of the screen and not show any output upon a successful connection. If for some reason you do want to use Putty, then follow the directions below

In Windows 7, Telnet is not installed by default so we need to turn it on, which we do from the Windows Features settings. Find this setting's page by simply searching for "Windows features" from the start menu search box.



Then tick "Telnet Client".




Click OK, let Windows do its thing, and then open a command window.

Watching HTTP work

We're going to communicate directly with the Web Server and "speak" HTTP to it.

We'll use the test page we created earlier. Point your Web Browser at localhost and make sure that page comes up.

**LEARNING ACTIVITIES**

ACTIVITY 7

HTTP

Now we're ready to see HTTP work at a very low level, which will help us understand what's happening when we're using developer tools later. Connect to your Apache Server (the "A" in WAMP) on localhost, on port 80 using either Putty or Telnet and issue this:

```
GET /
```

Hit the enter key twice after the slash.

What do you get? Something like this hopefully:

```
<html>
<body>
    Hello from the Server
</body>
</html>
```

Connection Closure

The HTTP connection closes immediately. That text is the Server's response, which just so happens to be a document in HTML format.

You'll immediately get a dialog message from your terminal client saying something like "Connection has been closed". That means the Server hung up on us.

Note that due to an unfortunate bug with Putty, the title bar always says "inactive" no matter what the state of the connection is.



HTTP protocol

Now we will conform to more of the HTTP protocol to create a more legitimate connection. Restart the connection (using the title bar menu in Putty) and issue:

```
GET / HTTP/1.1
Host: localhost
```

Again, hit the enter key twice after the last part. You should get something like:

```
HTTP/1.1 200 OK
Date: Thu, 27 Feb 2014 03:14:39 GMT
Server: Apache/2.4.4 (Win32) PHP/5.4.16
Last-Modified: Thu, 27 Feb 2014 03:03:43 GMT
Accept-Ranges: bytes
Content-Length: 58
Content-Type: text/html

<html>
<body>
    Hello from the Server
</body>
</html>
```

Interpreting the response

So the Web Server has said "Oh, you speak HTML 1.1? Let me respond accordingly with extra information". All that extra information before the content is:

- > Response code: 200 which means "success"
- > Header information regarding the responder (Server) and the response.

Investigate the other response codes at: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html> and <https://urivalet.com/reason-phrases/>

Should this link be unavailable please report the issue to TAFENow and instead search the internet for "Status Code and Reason Phrase".


Note that there was a new requirement on the part of our valid HTTP 1.1 request - we also needed to form a header, in this case all that was required was announcing who we were as the "host".

Did you notice that the connection didn't close immediately? It stays open for some amount of time set by the Server. This is the Server deciding that, since you're a HTTP 1.1 compatible client, it will keep the connection open so you can make other GET requests before the connection is closed.

If you're fast enough it is possible to making another request:

```
GET / HTTP/1.1
Host: localhost
```

You should get the page again.

	LEARNING ACTIVITIES	ACTIVITY 9
<p>“Keep Alive”</p> <p>Locate the <code>httpd.conf</code> file in your WAMP release and find the section that controls the connection “Keep Alive” feature.</p> <pre># KeepAlive: Whether or not to allow persistent connections (more than # one request per connection). Set to "Off" to deactivate. # KeepAlive On # # MaxKeepAliveRequests: The maximum number of requests to allow # during a persistent connection. Set to 0 to allow an unlimited amount. # We recommend you leave this number high, for maximum performance. # MaxKeepAliveRequests 100 # # KeepAliveTimeout: Number of seconds to wait for the next request from # the same client on the same connection. # KeepAliveTimeout 5</pre> <p>Change the <code>KeepAliveTimeout</code> to 60, which will give you more time to follow up with additional GET requests while you are playing around. Restart WAMP for the change to take effect.</p>		

Tools to examine open connections

It's worth taking the time at this point to examine the state of connections our machine is making to the outside world. This gives us an insight into the HTTP connections and their behaviour.

In Windows: you'll need to run cmd.exe as the Admin user. The easiest way to do this is to right click on cmd.exe in the start menu and click "Run as administrator". Then run `netstat -n -o -b 1`.

In Linux: `lsof -i -n` (or `netstat`)

On a test system, after browsing to Wikipedia, the output of `lsof` looked like this:

```
firefox  TCP localhost:52672->text-lb.ulsfo.wikimedia.org
(ESTABLISHED)
firefox  TCP localhost:46483->static.softlayer.com (ESTABLISHED)
firefox  TCP localhost:60242->bits-lb.ulsfo.wikimedia.org
(ESTABLISHED)
firefox  TCP localhost:52592->upload.ulsfo.wikimedia.org
(ESTABLISHED)
```

As we can see, a single Wikipedia page causes the browser to make connections to several different Servers as it gathers the resources to display a page – Wikipedia needs to spread its resources around to different Servers because of how popular the site is ("lb" probably stands for "load balance") After a few seconds, these connections all close:

```
firefox  TCP localhost:52672->text-lb.ulsfo.wikimedia.org
(CLOSE_WAIT)
firefox  TCP localhost:46483->static.softlayer.com (CLOSE_WAIT)
firefox  TCP localhost:60242->bits-lb.ulsfo.wikimedia.org
(CLOSE_WAIT)
firefox  TCP localhost:52592->upload.ulsfo.wikimedia.org
(CLOSE_WAIT)
```

On a Windows machine we see these open connections:

```
TCP localhost:49168 173.194.72.125:5222 ESTABLISHED [chrome.exe]
TCP localhost:49261 54.72.21.169:80 ESTABLISHED
[Avira.ServiceHost.exe]
TCP localhost:57558 103.10.4.40:21 ESTABLISHED [AI Suite II.exe]
```

That connection from Chrome stays open because the user is listening to music online. Avira the anti-virus program has an open connection – not much to say about that because if we don't trust our anti-virus program we have big problems. But what's that "AI Suite II.exe"? That's the ASUS motherboard's utility program. Why does it have an internet connection open all the time? This might demand investigating!¹

¹ If you're interested in delving even deeper into HTTP and networking, check out WireShark



Open connections

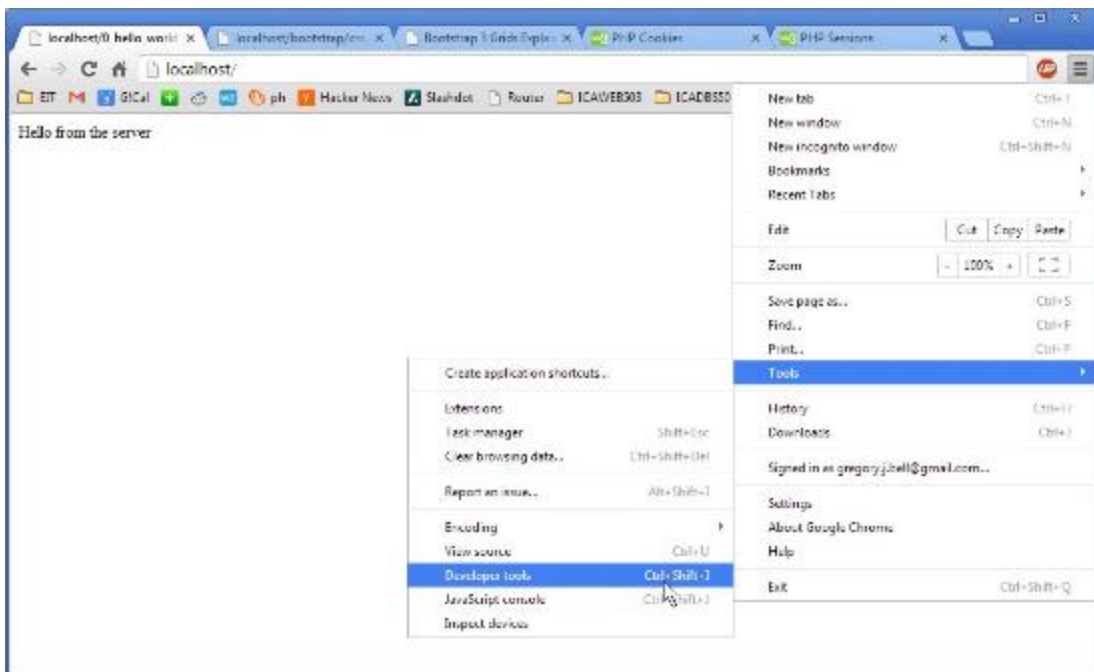
Re-execute the GET command we used above and re-examine the list of open connections. Do you see a new one from your Telnet client now? Does it close after 60 seconds?

Browser-based developer tools

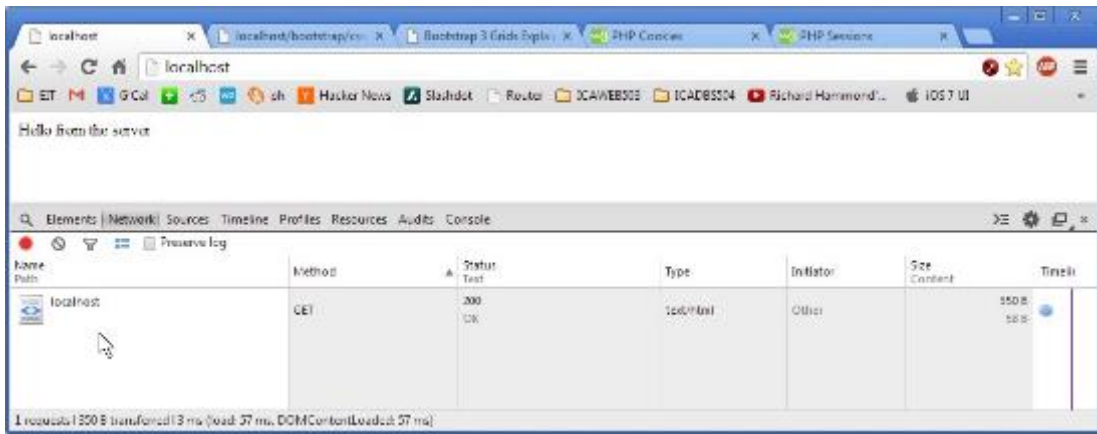
All the major browsers have “developer tools” built into them to allow you to inspect the HTTP requests and responses with a more user-friendly interface. Now that we’ve seen how the communication happens at a low level, we’re in a better position to make sense of what these tools show us.

These instructions and screenshots come from Google’s Chrome browser.

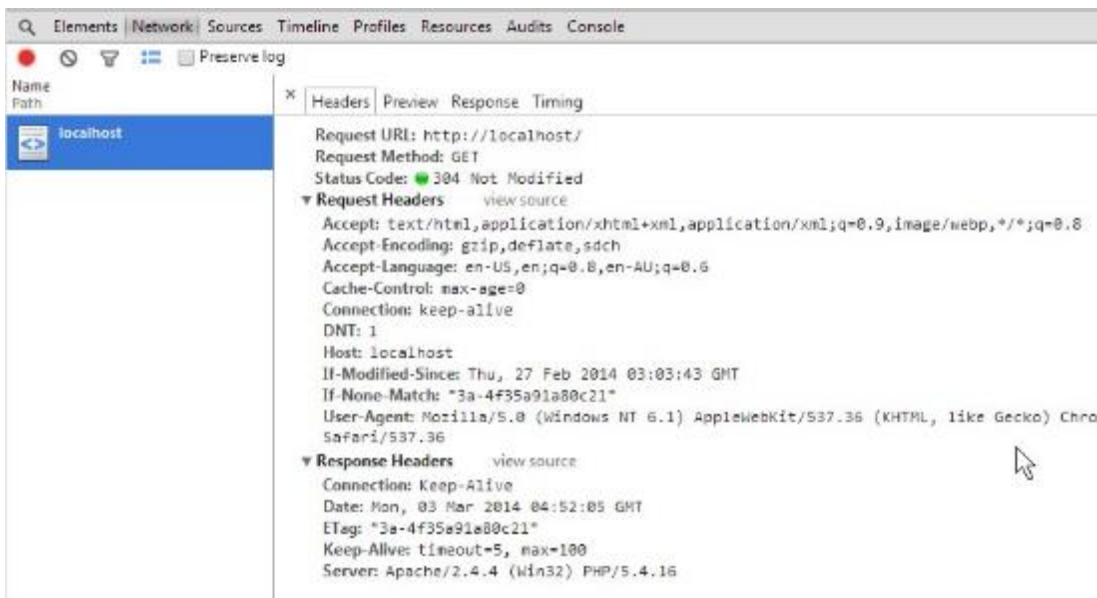
First, enable the Developer Tools Window, use Ctrl-Shift-I (right-click to Inspect Element) or go through the menu:



Click on the Network tab and reload the page. You’ll get a nice display of the resources requested and how long it took.



The resource name in the left column is a link. Click on it.



Those are the headers – the options and data that went along with the request and response. We can see the “Host:” header that we manually included before. The “Response Headers” are the headers the Server sent back.


Click on “view source” next to the “Request Headers”.

▼ **Request Headers** view parsed

```
GET / HTTP/1.1
Host: localhost
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.101 Safari/537.36
DNT: 1
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,en-AU;q=0.6
If-None-Match: "3a-4f35a91a80c21"
If-Modified-Since: Thu, 27 Feb 2014 03:03:43 GMT
```

This shows us the raw data sent, with the biggest difference from the pretty parsed version being that we can see the `GET / HTTP/1.1` – just like we sent from the terminal program!

Lastly, click on the response tab. Here we can see the data part of the response. In this case, it was simple HTML.

	LEARNING ACTIVITIES	ACTIVITY 11
<p>Google Chrome</p> <p>If you do not already have Google Chrome install it. Complete the steps that are covered in this section.</p> <p>Chrome’s developer tools are quite amazing. Take a look around and see if you can find anything else that might be useful to you. Note: there will be some additional features covered later in this unit.</p>		

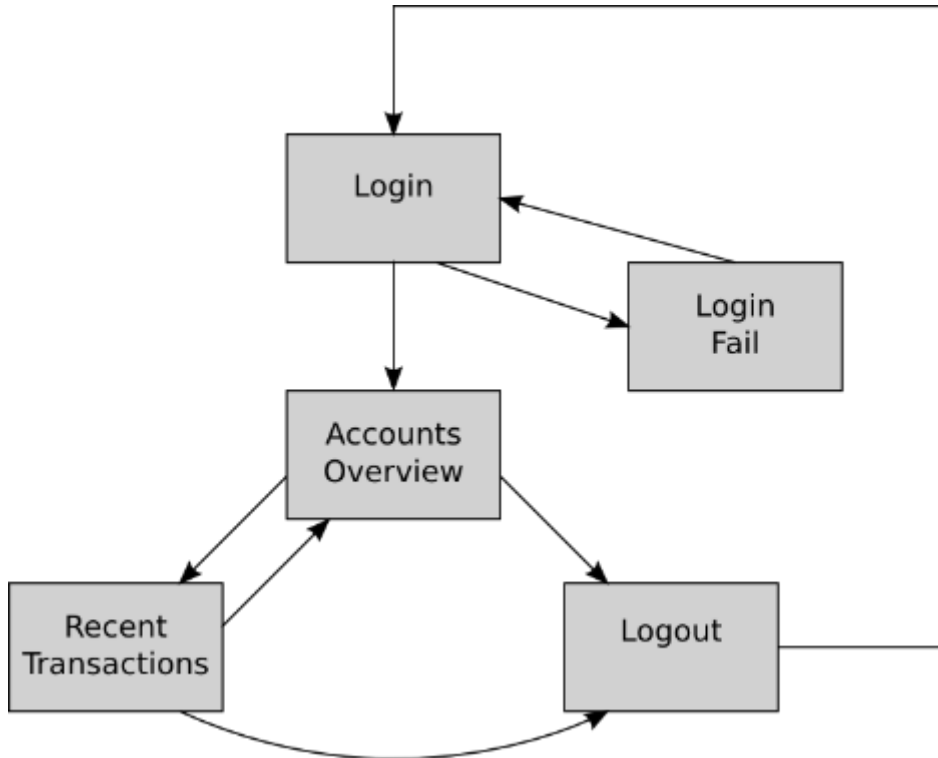
The Stateless web

Before we understand “stateless”, we’d better understand what “state” is. It’s defined as “the condition that someone or something is in at a specific time”. Many systems, both hardware and software, maintain a current state, which allows them to respond depending not just on the current input, but also based on the current state.

State machines

A state machine is a set of states and transitions. One state is active at a time. The active state is determined by previous states and current inputs.

Figure 1 – A typical state machine



This model is so useful that shows up in many places, including Web Applications. State machines can be implemented in hardware or software.

A key feature of state machines in the next state depends on the current state, as well as inputs. For example, in the above diagram, the “Recent Transactions” state can only be arrived at when the machine is in the “Accounts Overview” state.

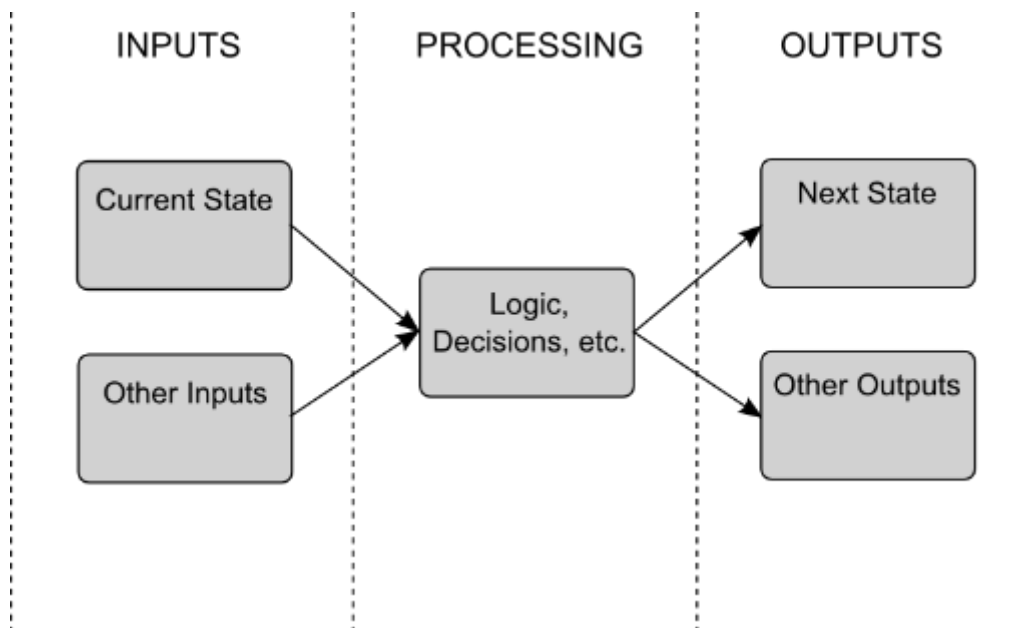
How does the machine “know” its current state? It must remember this information. Thus, state machines have memory of the current state.

You might notice that the above picture looks a lot like pages on a website, with the links between pages representing the transitions between states of the state machine. This is accurate, and in this case the Web Browser represents the memory in the system.

However, a Web Application usually wants to provide a richer experience, remembering things like items in a shopping cart, user names, or other more complicated state information that is not as easily represented simply by the page that the user is on or requesting.

We might draw a diagram relating to the inputs and outputs in a state machine as follows:

Figure 2 – Inputs and outputs in a state machine



State and sessions

What's interesting about your call to the bank is that the bank has no idea that it's "you" calling again. This is what we mean by communications over HTTP being "stateless". The Server has no idea what state your session with it is in. Did you login? Have you been here before? Did you start from the homepage?

So a session with a website may last many hours, or even days or weeks. You may log in to check some flights and plan a trip, then wander off to get a coffee.

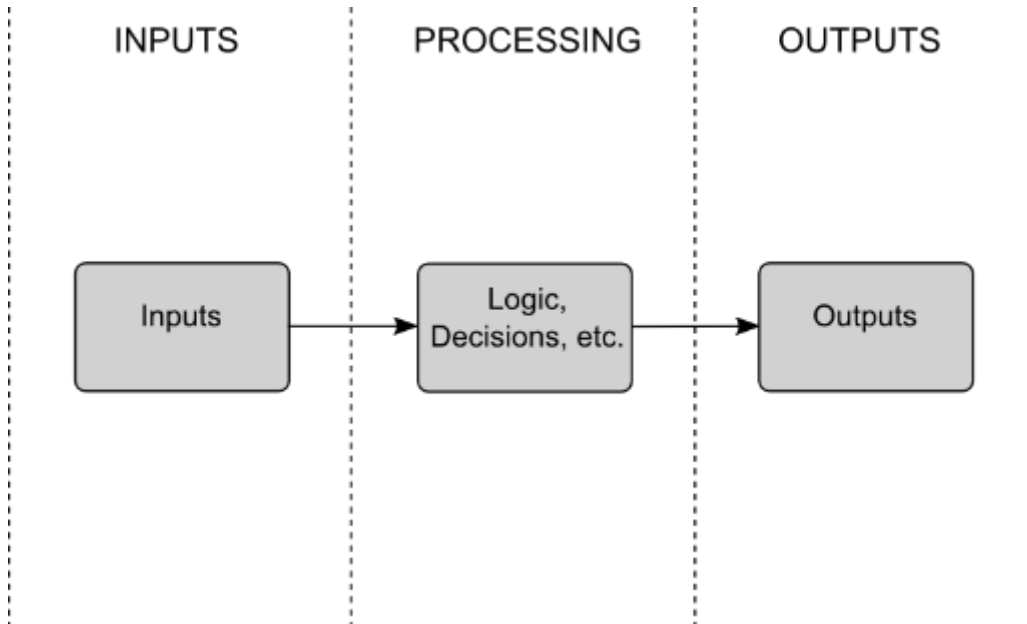
We would call this conversation you're having with the bank or travel website a "session". It's made up of several communications, but only you and the bank really know when it's over – when you close the window, when two weeks have gone by, or when you click "Logout".

Sessions require state information.

Stateless systems

In contrast to the above diagram, a stateless system would not have the current state as an input:

Figure 3 – A stateless system



In a stateless system, operations are completely detached from operations that have come before, and those that come next. Usually a stateless system is simpler, and since it doesn't require memory to keep track of current state, it requires fewer resources.

HTTP is stateless

Now that we understand what states are, let's look at what web technologies bring to the table.

In the case of HTTP, there's no definition of any sort of retention of state. This is for resource frugality – keeping track of state by “keeping the line open” would be costly in terms of memory and network resources.

An analogy with our phone call example would work. There would be two ways the bank could remember you – one would be to keep the phone call going forever, so that it was all one conversation. Thus, the poor bank representative on the other end would remember who you were and what you had spoken about. Since HTTP “hangs up” after every page, this doesn't happen.

The other way for there to be memory or “state” on subsequent calls to the bank would be if they would look up your details based on some information you send them. This is in fact how the bank knows you. And, while HTTP doesn’t implement this, we’ll see later that this is exactly how Web Applications provide useful behaviour that statefulness allows.

PHP is stateless

What about our program running on the Web Server? A PHP program runs and terminates entirely in a single GET request (note that this is covered in ICTDBS504). There is no memory retained from one run of the program to the next, so there is no state retained from one Server request to the next. In practice, this means that each time the PHP program “wakes up” it must be told exactly what to do by the request.

In other words, when your browser does a GET request, everything that defines what is needed must be in that request. The Server has no idea who you are, that you just authenticated, what your preferences are, etc.

Stateless systems and statefulness

So the lack of state in HTTP and PHP is a problem. When you interact with a website, you expect it to remember certain things, like:

- > your name
- > what’s in your shopping cart
- > the fact that you are authenticated
- > the details of your currently planned trip

What we will need is something built on top of HTTP and/or PHP which gives us statefulness.

There is another example of this in the web network stack – IP is a stateless protocol, and the TCP protocol is stateful. This is why we always refer to the Internet protocol as TCP/IP. Both are needed.

Let’s create a simple Web Application that has no knowledge of state. We need two files.

Index.php looks like this:

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>Stateless web app</title>
6 <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
7 <link href="/bootstrap/css/bootstrap-theme.css"
  rel="stylesheet">
8 <link href="/style.css" rel="stylesheet">
```

```

9
10 </head>
11
12 <body>
13 <br/>
14 <div class="col-sm-4">
15 <h1>I don't know you</h1><br/>
16 <form method="post" action="form_process.php">
17   And you are?<br/>
18   <input type="text" name="name"><br/>
19   Please tick the foods you like:<br/>
20   <input type="checkbox" name="food" value="apple">Apple<br/>
21   <input type="checkbox" name="food"
   value="chocolate">Chocolate<br/>
22   <input type="checkbox" name="food" value="liver">Liver<br/>
23   <input type="checkbox" name="food" value="pizza">Pizza<br/>
24   <input type="submit" value="Save" class="btn btn-primary">
25 </form>
26 </div>
27 </body>
28 </html>

```

This is no bare-bones HTML file – it uses the Bootstrap CSS framework (lines 6-7), as well as a local style sheet (line 8). If you're not familiar with Bootstrap, it's covered a bit in ICTDBS504, but you can simply download it from <http://v4-alpha.getbootstrap.com/getting-started/download/>.

Should this link be unavailable please report the issue to TAFENow and instead search the internet for "Bootstrap download".

To install it unzip it to C:\wamp\www.

The file also implements a simple form. The form is processed with:

form_process.php:

```

1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>Stateless web app</title>
6 <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
7 <link href="/bootstrap/css/bootstrap-theme.css"
   rel="stylesheet">
8 </head>
9
10 <body>
11 <h2>
12 Right, so you are <?php echo $_POST['name'] ?>. Got it, Boss
   (not really)</h2>
13 <a href="/">Return to front page</a>
14 </body>
15 </html>

```

Since this code has no way of storing the data, it can only echo it (line 12). When you return to the front page, it has completely forgotten about you. Try it.

The style.css file looks like this:

```
1  body {
2    padding: 20px;
3  }
4
5  input, input[type="checkbox"] {
6    margin: 5px;
7  }
8
9  div {
10   margin: 20px;
11 }
```



LEARNING ACTIVITIES

ACTIVITY 12

POST data

Use the developer tools to inspect the POST data sent when you click the “Save” button on the form. You can see it from the Headers tab under “Form Data”.

We’ve manually communicated with a HTTP Server and learned a bit about the protocol and what headers do. We’ve learned how to see connections being made from our local machine to the outside world and we’ve learned about state machines and how HTTP and PHP are inherently stateless with a simple forgetful Web Application – paving the way for some technologies and techniques we’ll learn about in the next topic.



Topic 3 – The Stateful web

So what will we put on top of HTTP and PHP in order to provide the state information that is required for the modern web experience?

URLs as storage

One way to provide state information to the Server, and specifically the PHP code running on the Server, is to encode information in the URL.

Let's modify `index.php` to do this:

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>Stateful web app</title>
6 <link href="/style.css" rel="stylesheet">
7 <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
```

```

8     <link href="/bootstrap/css/bootstrap-theme.css"
      rel="stylesheet">
9 </head>
10
11 <?php
12 function checkbox ($food, $foods) {
13     print "<input type='checkbox' name='foods[]' value='";
14     print $food . "' ";
15     print in_array($food, $foods) ? "checked" : "";
16     print ">" . ucwords($food) . "<br/>";
17 }
18 ?>
19
20 <body>
21 <br/>
22 <div class="col-md-4">
23 <form method="post" action="/form_process.php">
24 <?php
25 if (isset($_GET['name'])):
26     print "<pre>";
27     print_r($_GET);
28     print "</pre>";
29     $foods = $_GET['foods'];
30     $name = $_GET['name'];
31 ?>
32 <h1>Welcome back, <?php print $name ?></h1>
33 <input type="hidden" name="name" value="<?php print $name;
34 ?>">
35 <?php else: ?>
36 <h1>I don't know you</h1><br/>
37 And you are?<br/>
38 <input type="edit" name="name"><br/>
39 <?php $foods = []; ?>
40
41 <?php endif ?>
42
43 <fieldset>
44 <legend>Please tick the foods you like:</legend>
45 <?php checkbox("apple", $foods);?>
46 <?php checkbox("chocolate", $foods);?>
47 <?php checkbox("liver", $foods);?>
48 <?php checkbox("pizza", $foods);?>
49 </fieldset>
50 <input type="submit" value="Save" class="btn btn-primary">
51 </form>
52 </div>
53 </body>
54 </html>

```

There's a lot that's changed. First, we look at the `_GET` variable to see if we were passed any information in the URL (line 25). We then print debug statements (lines 26-28) and populate our local variables (lines 29-30).

We then welcome back the user because we know who they are. Note that we also need input of type "hidden" (line 33) in order to pass this data back to the form again when the user hits the save button. The name is not saved anywhere else so must be saved in this form, even though it is not displayed.

If we were not passed any information about this person or their preferences, we prompt them for their name as before and create an empty foods array (lines 35-39).


Side note: Somewhat unnaturally, we've moved the form tag to the top of the file (line 23), or the h1 tag inside the form, depending on how you look at it. This prevents us repeating the form tag in each branch of the 'if' statement. When possible, don't repeat yourself.

Then we generate the check boxes, but this time using a function (lines 12-17) to avoid repeating code. The function checks for the existence of the food type in the foods array, and sets the checkbox appropriately. Viewing the generated source code in the browser (View Source) will help make it clearer what this function does.

Now make `forms_process.php` like this:

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>Stateful web app</title>
6 <link href="/style.css" rel="stylesheet">
7 <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
8 <link href="/bootstrap/css/bootstrap-theme.css"
  rel="stylesheet">
9 </head>
10
11 <body>
12 <?php
13     print "<pre>";
14     print_r($_POST);
15     print "</pre>";
16 ?>
17 <h2>
18     Right, so you are <?php echo $_POST['name'] ?>. Got it, Boss
19     (really this time)</h2>
20 <?php
21     $url = http_build_query($_POST, '', '&');
22     print "<a href='/?' . $url . "'>Return to front page</a>"
23 ?>
24
25 </body>
26 </html>
```

The key to this page is on lines 21-22, where we're using the PHP function `http_build_query` to build a URL from the posted data. This URL is then used in the "Return to front page" link. This is how we are encoding the user data into the URL.

	LEARNING ACTIVITIES	ACTIVITY 13
<p>Updates</p> <p>Proceed to make the updates described above in the code you are working with. Hover over the link that is being built and observe how it's not just a straightforward link to /.</p> <p>Have you seen messy URLs like this before?</p> <p>Now click on the "Return to front page" link and observe how the front page now knows who you are and what your preferences are. Easy, right?</p> <p>Also notice that the URL shown in the browser's address bar is now:</p> <p>http://localhost/?name=Fred&foods%5B0%5D=apple&foods%5B1%5D=chocolate</p>		

Advantages and disadvantages of this approach

You can see all the state information passed from `form_process.php` right there in the URL. This is both an advantage and a disadvantage. On one hand the data goes with the link as if it is emailed or bookmarked. On the other, it's easy to see what's being tracked and for the user to change it, possibly exploiting a security hole. Another disadvantage of this technique is that it requires the Server to modify the links in the page it's outputting, to reflect the current state. This is sort of messy. We'll look at other disadvantages when we look at security.

Note: If you have any problem reading the code above, your three best sources available at the time of publication are:

<http://www.w3schools.com/tags/>

<http://stackoverflow.com/>

and

<http://www.php.net/manual/en>

You might consider using the Google Custom Search feature to create a customised search engine that limits your search results to just those three sources (and any others that you like).

Cookies as storage

To support state for Web Applications, HTTP and Web Browsers work to support data stored local to the user called "cookies". The Server sets a cookie as part of its response to the browser, and the browser stores this data and sends it back to the Server with every GET request from then on.

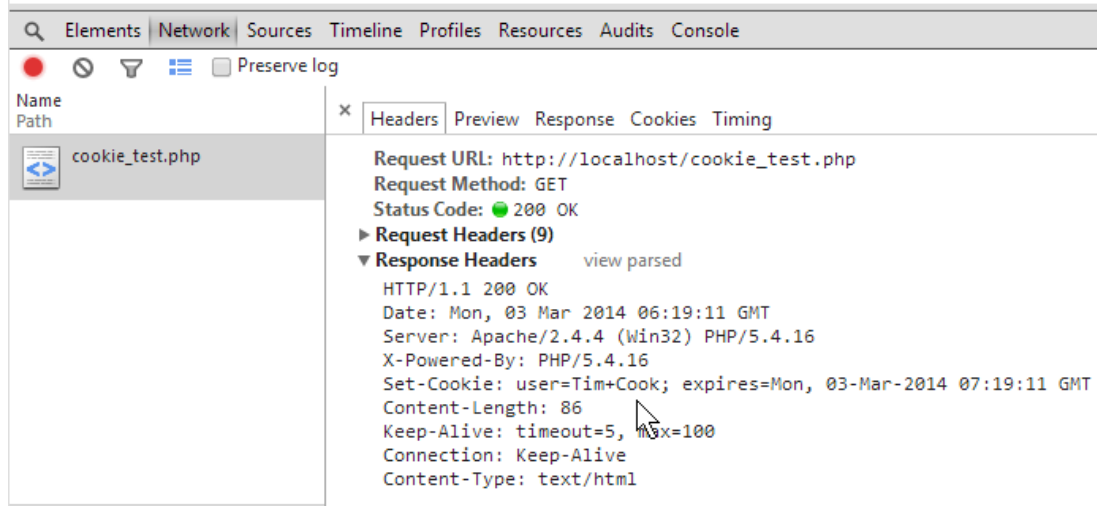
Let's see this work.

`cookie_test.php`:

```
1  <?php
2  setcookie("user", "Tim Cook", time()+3600);
3  ?>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  </head>
8  <body>
9    <h2>I set a cookie</h2>
10 </body>
11 </html>
```

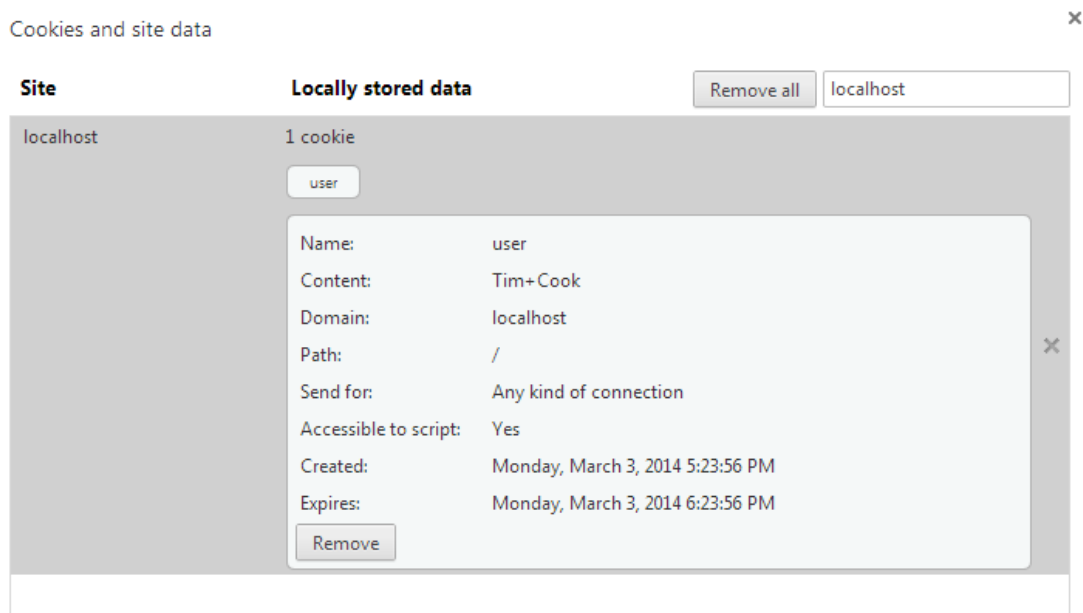

Open Developer Tools to the Network tab and point your browser to `http://localhost/cookie_test.php`. Doesn't look like much happened. But let's look at what the Server sent us:

I set a cookie



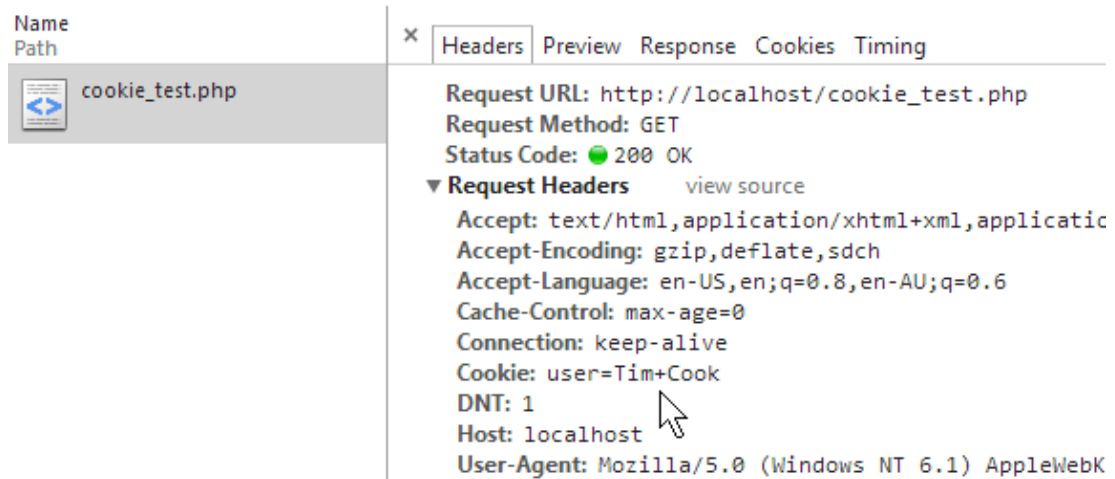
Look at the Set-Cookie header. The cookie has a variable "user", with a value of "Tim Cook", and it expires 3600 seconds from now.

Let's look in the browser for this cookie (in Chrome, this is under Settings, Privacy, Content Settings, All Cookies and Site Data).



Makes sense, right? Until the expiry time, the browser will send the data in that cookie to the website 'localhost' on every GET.

Let's look at a GET now that the cookie has been set. Refresh the page and look at the Request Headers. Remember, this is data from the browser sent to the Server.



So cookies are a way of getting state information to the Server so that it can give us a customised experience.

Note that since cookies are sent in the header, the **set_cookie** function call needs to be above all HTML.²

Our more sophisticated example

Here we will consider what happens when we upgrade the original example to use cookies.

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <title>Stateful web app</title>
6 <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
7 <link href="/bootstrap/css/bootstrap-theme.css"
  rel="stylesheet">
8 <link href="/style.css" rel="stylesheet">
9 </head>
10 <?php
11 function checkbox ($food, $foods) {
12     print "<input type='checkbox' name='foods[]' value='";
13     print $food . "' ";
```

² Interestingly, the author didn't do this at first, but the cookie was still set in the header. Something in PHP or Apache was ordering the output so that the headers were always first! Without understanding what was doing this, it's safest to control the output manually by putting all **set_cookie** calls before the HTML. This is considered best practice anyway.

```

14     print in_array($food, $foods) ? "checked" : "";
15     print ">" . ucwords($food) . "<br/>";
16 }
17 ?>
18 <body>
19     <br/>
20     <div class="col-md-8">
21         <form method="post" action="/form_process.php">
22             <?php
23                 if (isset($_COOKIE['name'])):
24                     $name = $_COOKIE['name'];
25                 else:
26                     $name = '';
27                 endif;
28
29                 if (isset($_COOKIE['foods'])):
30                     $foods = unserialize($_COOKIE['foods']);
31                 ?>
32                 <h1>Welcome back, <?php print $name; ?></h1>
33                 <input type="hidden" name="name" value="<?php print $name;
34                 ?>"><br/>
35             <?php else:
36                 $foods = [];
37             ?>
38             <h1>I don't know you</h1><br/>
39             And you are?<br/>
40             <input type="edit" name="name"><br/>
41
42             <?php endif; ?>
43
44             <fieldset>
45                 <legend>Please tick the foods you like:</legend>
46                 <?php checkbox("apple", $foods);?>
47                 <?php checkbox("chocolate", $foods);?>
48                 <?php checkbox("liver", $foods);?>
49                 <?php checkbox("pizza", $foods);?>
50             </fieldset>
51             <input type="submit" value="Save" class="btn btn-primary">
52
53         </form>
54         <div class="col-md-8 alert alert-info">
55             Debug output:<br>
56             <?php print_r($_COOKIE);?>
57         </div>
58     </div>
59 </body>
60 </html>

```

You can see most of the change involves just fetching state data from `$_COOKIE` instead of `$_GET`. To get the foods array into and out of the cookie, we've employed PHP's `serialize` and `unserialize` functions, as can be seen in `form_process.php`:

```

1     <?php
2
3         if (isset($_POST['foods'])) {
4             $foods = $_POST['foods'];
5         } else {

```

```

5         $foods = [];
6     }
7     setcookie('foods', serialize($foods), time()+3600);
8     setcookie('name', $_POST['name'], time()+3600);
9     ?>
10    <!DOCTYPE html>
11    <html>
12
13    <head>
14        <title>Stateless web app</title>
15        <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
16        <link href="/bootstrap/css/bootstrap-theme.css"
17            rel="stylesheet">
18        <link href="/style.css" rel="stylesheet">
19    </head>
20    <body>
21        <h2>
22            Right, so you are <?php echo $_POST['name'] ?>. Got it, Boss
23            (with cookies this time)</h2>
24        <a href="/">Return to front page</a><br>
25        <div class="alert alert-info">
26            Debug output:<br>
27            <?php print_r($_POST);?>
28        </div>
29    </body>
30    </html>

```

We must create and set the cookie (lines 1-9) before any HTML code since the cookie setting happens with an HTTP header. Headers must come before the response data.

There are a few steps involved because if the user hasn't checked any checkboxes, we do not get sent an empty foods array – we get sent no foods array. So we must check for its existence and create an empty one if needed (lines 1-9).

Lines 7 and 8 are where the important stuff happens – the saving of the user's data. Notice we have a plain URL for the front page again (Line 23).

We've also changed the debug output (lines 25-28) to take advantage of some Bootstrap features, just to try a different look.

```

Debug output:
Array ( [name] => Fred [foods] => a:4:
  {i:0;s:5:"apple";i:1;s:9:"chocolate";i:2;s:5:"liver";i:3;s:5:"pizza";}
)

```

That shows pretty nicely what the serialize function does. It turns our array into a string. That string describes the array:

a : 4 expect an array of 4 elements

i : 0 first array element, key is an integer of value zero

s : 5 : "apple" value is a string of 5 characters which is "apple"

i : 1 second array element, key is an integer of value one

s : 9 : "chocolate" value is a string of 9 characters which is "chocolate"



LEARNING ACTIVITIES

ACTIVITY 14

Update your files

Go ahead and update your files to include cookies as detailed above and see how this all works for you. Take note of the various key points highlighted in the sophisticated example above.



LEARNING ACTIVITIES

ACTIVITY 15

Expiry time

Leave the browser on the home page, and refresh it in an hour. Notice that it forgets about you and your foods. This is because of the expiry set on the cookie when it was set (3600 seconds).

Sessions and cookies

Sessions are a feature in PHP which uses cookies to store a unique ID, which is then paired with user data stored on the Server. This is the major difference, but there are others.

Table 1 Cookies and sessions

Cookies	Sessions
Stored on client's computer	Stored on Server
Limited in size	Unlimited in size
Expiration controlled by browser	Expiration controlled by Server

Cookies	Sessions
Lives through browser exit	Destroyed on browser exit
Data seen by user*	Data hidden from user
Contents can be changed by the user*	Contents cannot be changed by the user
Must be sent every browser request	Only the Session ID is sent

The points with asterisks are major security concerns. If the user can see what's being stored, and change it before it's sent back, what happens if they change the user ID? Or the "logged in" status?

Again, sessions still use a cookie, but just to store a unique session ID in it.

Other cookies can be set to get the best of both worlds – for example, storing the user's login ID as a cookie, which will persist through browser runs and allow the site to recognise the user every time.

Let's see how sessions work with a simple example:

```

1  <!DOCTYPE html>
2
3  <?php
4
5  if(!isset($_SESSION)) {
6      session_start();
7  }
8
9  if (isset($_SESSION['name'])) {
10     $name = $_SESSION['name'];
11 } else {
12     $name = "Stranger";
13 }
14
15 ?>
16
17 <html>
18 <head>
19 </head>
20 <body>
21     <h2>Hello <?php echo $name;?></h2>
22     <form method="post" action="save_user.php">
23         <input type="text" name="name">
24         <button type="submit">Save name</button>
25     </form>
26 </body>
27 </html>

```

Again notice how we've put the session handling code above the HTML, since it all happens in the headers. We start the session with `session_start()`, if it's not apparent that it already exists (lines 5-7) and pull data out of it if it does exist (lines 9-13).

And here's `save_user.php`:

```
28 <?php
29
30 if(!isset($_SESSION)) {
31     session_start();
32 }
33
34 $_SESSION['name'] = $_POST['name'];
35
36 ?>
37
38 <!DOCTYPE html>
39 <html>
40 <head>
41 </head>
42 <body>
43
44 <?php echo "Saved " . $_POST['name'] . " as user name in a
    session."; ?>
45
46 </body>
47 </html>
```

A `$_SESSION` global is there for data storage and retrieval.

Open Developer Tools, Network tab, and point your browser at localhost and see the behaviour of the program.

On first load of this page, the Server will detect that there's no current session and set a session up in the response:

```
Set-Cookie:
PHPSESSID=r7ecn996qbcupadt1hf32cggc7; path=/
```

Then, in the `save_user` form, we can see the session ID cookie get sent to the Server by the browser in the request:

```
Cookie:
name=Fred;
foods=a%3A2%3A%7Bi%3A0%3Bs%3A5%3A%22apple%22%3Bi%3A1%3Bs%3A9%3A%22chocolat
e%22%3B%7D; PHPSESSID=r7ecn996qbcupadt1hf32cggc7
```

Wait a minute... what's all that extra cookie data? Well, this is the same browser that has been used for the other exercises; our other examples set cookies too. They're still there, and the browser is obligated to send them too, with every GET request. This shows one advantage of sessions – the PHPSESSID is usually shorter than the data you're wishing to store, saving bandwidth.

We will work through the updates of our application, to use sessions. Here's index.php:

```
1 <?php
2 function checkbox ($food, $foods) {
3     print "<input type='checkbox' name='foods[]' value='";
4     print $food . "' ";
5     print in_array($food, $foods) ? "checked" : "";
6     print ">" . ucwords($food) . "<br/>";
7 }
8
9 if(!isset($_SESSION)) {
10     session_start();
11 }
12
13 if (isset($_SESSION['name'])) {
14     $name = $_SESSION['name'];
15 }
16
17 if (isset($_SESSION['foods'])) {
18     $foods = unserialize($_SESSION['foods']);
19 } else {
20     $foods = [];
21 }
22
23 ?>
24
25 <!DOCTYPE html>
26 <html>
27
28 <head>
29     <title>Stateful web app</title>
30     <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
31     <link href="/bootstrap/css/bootstrap-theme.css"
32     rel="stylesheet">
33     <link href="/style.css" rel="stylesheet">
34 </head>
35 <body>
36     <br/>
37     <div class="col-md-4">
38         <form method="post" action="/form_process.php">
39             <?php
40                 if (isset($name)):
41                     ?>
42                     <h1>Welcome back, <?php print $name; ?></h1>
43                     <input type="hidden" name="name" value="<?php print
44                     $name; ?>"><br/>
45                     <?php else: ?>
46                     <h1>I don't know you</h1><br/>
47                     And you are?<br/>
48                     <input type="edit" name="name"><br/>
49                 <?php endif ?>
50             </div>
51 </body>
52 </html>
```



```

49     <legend>Please tick the foods you like:</legend>
50     <?php checkbox("apple", $foods);?>
51     <?php checkbox("chocolate", $foods);?>
52     <?php checkbox("liver", $foods);?>
53     <?php checkbox("pizza", $foods);?>
54     </fieldset>
55     <input type="submit" value="Save" class="btn btn-primary">
56 </form>
57 </div>
58 <div class="col-md-8 alert alert-info">
59 Debug output:</br>
60 <?php print_r($_SESSION);?>
61 </div>
62 </body>
63 </html>

```

Just for the sake of order, we've put all of our PHP logic up at the top, since the session start needs to be ahead of the HTML anyway.

The code looks much the same as when we were using `$_COOKIE`.

Study the code.

Now here's `form_process.php`:

```

1  <?php
2  if(!isset($_SESSION)) {
3      session_start();
4  }
5  ?>
6  <!DOCTYPE html>
7  <html>
8
9  <head>
10 <title>Stateful web app</title>
11 <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
12 <link href="/bootstrap/css/bootstrap-theme.css"
13     rel="stylesheet">
14 <link href="/style.css" rel="stylesheet">
15 </head>
16 <body>
17 <h2>
18 Right, so you are <?php echo $_POST['name'] ?>. Got it, Boss
19 (with sessions this time)</h2>
20 <?php
21     if (isset($_POST['foods'])) {
22         $foods = $_POST['foods'];
23     } else {
24         $foods = [];
25     }
26     $_SESSION['foods'] = serialize($foods);
27     $_SESSION['name'] = $_POST['name'];
28     print "<a href='/'>Return to front page</a></br>";
29 ?>
30 <br/>
31 <div class="alert alert-info">

```

```
32     Debug output:</br>
33     <?php print_r($_POST);?>
34     </div>
35 </body>
36 </html>
```



LEARNING ACTIVITIES

ACTIVITY 16

Sessions

Go ahead and update the code you have been working on to implement sessions as detailed above. Use the app, study the data given by the developer tools, and make sure you understand how it all works.

Storage of session data

But wait a minute – we said the data was stored on the Server. But we didn't set up a database for PHP to use... so where does it store session data? It must be in a file! The file is in `C:\wamp\bin\php\php5.4.16\php.ini` (or `/etc/apache2/php.ini` or similar on Linux) and you'll see where.

```
session.save_path = "c:/wamp/tmp"
```

Well that's clear!



LEARNING ACTIVITIES

ACTIVITY 17

Find the data

Explore `C:\wamp\tmp` and see if you can find the data for your session. Use a text editor to examine the contents.

Session settings

While we're in `php.ini`, here are just a few more of the settings:

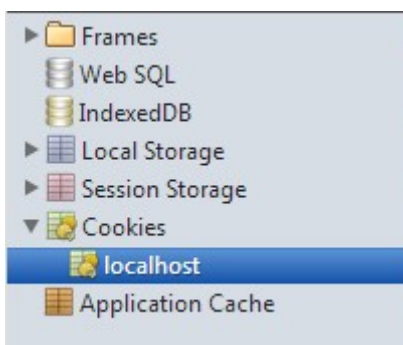
```
1  [Session]
2  session.save_handler = files
3  session.save_path = "c:/wamp/tmp"
4  session.use_cookies = 1
5  ;session.cookie_secure =
6  session.use_only_cookies = 1
7  session.name = PHPSESSID
8  session.auto_start = 0
9  session.cookie_lifetime = 0
10 session.cookie_path = /
11 session.cookie_domain =
12 session.cookie_httponly =
13 session.serialize_handler = php
```

And those are only the most interesting options. As you can see, sessions are a complex topic and you may be interested in dig in further. The actual file is chock full of comments documenting what these options do.

Instead of using cookies to store your session, PHP can use the URL as we investigated before. As before, there are advantages and dangers to having data in the URL. If the browser has cookies disabled, PHP will down-shift to using the URL, unless told not to with the options.

Other browser storage schemes

Other schemes for storing data and providing statefulness for Web Applications have been proposed. You can see some of them from the Resources tab in Chrome's Developer Tools.



Research a few of those and see if they're worth using in your next project.

Terminating sessions

At some point, the session must terminate. This could be from:

- > Server action
- > User action

By Server action

The Server may decide the session is over. This may be because of a timeout, a detected behaviour that looks like it may be a security issue, or any number of reasons.

By user action

Typically a user terminates a session by clicking a “Logout” button. Let’s look at implementing this in our Web Application.

Add this below to the existing form in `index.php`:

```
1     <form method="post" action="/session_destroy.php">
2         <input type="submit" value="Logout" class="btn btn-
3         danger"><br/>
4     </form>
5     And make session_destroy.php to handle when the user hits the
6     “Logout” button:
```

And make `session_destroy.php` to handle when the user hits the “Logout” button:

```
1  <?php
2      if(!isset($_SESSION)) {
3          session_start();
4      }
5      session_destroy();
6  ?>
7
8  <!DOCTYPE html>
9  <html>
10
11  <head>
12      <title>Stateless web app</title>
13      <link href="/bootstrap/css/bootstrap.css" rel="stylesheet">
14      <link href="/bootstrap/css/bootstrap-theme.css"
15      rel="stylesheet">
16      <link href="/style.css" rel="stylesheet">
17  </head>
18  <body>
19      <h2>I've already forgotten you</h2><br>
20      <a href="/">Return to front page</a>
21  </body>
22
23  </html>
```

**Debug output**

Everywhere you have debug output, replace the `print_r` with `var_dump` and see if you like this new type of debug output better.

Try out this modification and observe the changed behaviour. Also note that the session cookie doesn't get deleted. How does the Server know the session has been terminated then (hint: What other data is part of the session)?

We've added stateful behaviour to our application using three techniques – URL, cookies, and sessions. Each has its advantages and disadvantages. We've also continued to explore the Developer Tools built into Chrome (or IE or Firefox) to see what's happening under the bonnet – a skill which always comes in handy when things don't quite work right the first time.



Topic 4 – Security threats

Stateful applications involve storing data, and this data is probably sensitive, meaning security must be considered. If it's not sensitive, then at the very least it might be able to be used to trick the Server.

Sensitive information

Bank account numbers are a common piece of sensitive data. If this was stored in a cookie, and the user's computer was stolen, the thief would then know the user's bank account number.

User tinkering

Any data stored locally – in URLs or in cookies – can be manipulated by the user. If they do this, can they impersonate another user? Can they log themselves in even without supplying a password? Can they increase their credits in a game?

Tinkering in transit

Data from cookies can be manipulated between the user and the Server, to the usual effects of tricking, impersonating, or stealing, creating security

Internet security is, of course, a huge topic, but here are some ways to increase the security of the session data stored by Web Applications.

Encrypting

Connecting to a Server over an HTTPS encrypted connection is good practice and immediately alleviates many concerns.

Look up `session.cookie_secure` and understand what it does and how it works.

Signing

Data from the Server can be cryptographically signed, ensuring its validity and preventing tampering in between.

Hashing

By making the session IDs random, PHP automatically makes it hard to pretend to be someone else. Requiring an additional piece of matching, but unrelated, data is a good extension to the security of session IDs (e.g. user IP address, password, etc.)

Verifying IP


A Server could only accept session IDs from the originating Server. This may cause problems with people roaming between networks, or users on the other side of a router providing Network Address Translation ('NAT').


Spoofing / replay

One method of breaking security involves replaying the transactions between a client and Server, either to the Server or to the client. Some thought could be given to preventing this, for example by storing the current time in an encrypted cookie set at the same time as the session cookie, and confirming that they match.

Cookies-only

PHP has a setting preventing it from downshifting to using URL-based data storage in the event that cookies are not available (browser doesn't implement them, or the user has turned them off). The setting is `session.use_only_cookies` but be sure to read up on exactly what it does and what other settings are related to this behaviour. The downside of this is that your app may not work for some users.

	LEARNING ACTIVITIES	ACTIVITY 19
Sessions		
Look up <code>session.cookie_httponly</code> and <code>session.use_strict_mode</code> and in your own words write down what these do.		

	READINGS	RECOMMENDED 1
Read the following Wikipedia pages about session attacks.		
https://www.owasp.org/index.php/Session_Fixation		



Should this link be unavailable please report the issue to TAFENow and instead search the internet for "session fixation"

https://www.owasp.org/index.php/Session_hijacking_attack

Should this link be unavailable please report the issue to TAFENow and instead search the internet for "session hijacking"

No Web Application is ready to go "live" without the developer taking all appropriate precautions to secure user data and prevent unauthorised access, theft and trickery. We've looked at ways to attack a Web Application, and some defensive actions to take. The overall topic is much larger than addressed by this unit, but strong technical knowledge and a fair amount of paranoia is appropriate.

In this unit, we've looked at some interesting tools (Bootstrap, Telnet/Putty and browser built-in Developer Tools). We've learned low level details of the workhorse of the web – HTTP. We've learned how URLs, cookies and sessions can be used to store user data and provide memory to applications – or "state" as it's known.

You've also written some basic HTTP and PHP which implemented statefulness and forms.

You can extend this basic knowledge to begin to design sophisticated Web Applications that provide modern features people have come to expect.