How the Web Works

Before we can learn to write programs for the web, we need a better understanding of how the web works.

Static Webpages

Let's start out by taking a look at what happens when you type an address into your web browser for a static webpage—i.e., one which does not change in response to user inputs.

We enter a URL or Uniform Resource Locator like this into the web browser:

```
http://www.stanford.edu/about/history/
```

The web browser communicates with a web server requesting the resource found at the specified location. The web browser and the web server communicate using the HTTP or HyperText Transfer Protocol. We'll take a closer look at this protocol in a moment.

Typically the resource found is file written using HTML or HyperText Markup Language. This file specifies the text which will be displayed on the webpage and the structure of the text, such as where the headings and paragraphs are. It often specifies additional files which will be needed by the web browser to display the webpage. These files may include a CSS (Cascading Style Sheet) file used to provide formatting information. The HTML file will also specify the locations of any image files, sound files, or video files which are needed by the webpage. All these additional files will be transferred using the same HTTP protocol used by the original HTML file.

Dynamic Webpages

Webpages where the contents of the webpage changes in response to user inputs generally fall into one of two categories.

Server-Side Processing—In this approach, the user enters information into the webpage, typically using an HTML form. When they're done, they click on the submit button. The information is then transmitted to the web server using HTTP. The web server looks at the HTTP request and runs a program based on the information sent with the request. This program generates a new HTML file which is sent to the web browser.

Client-Side Processing—In this approach when the webpage is first sent to the user's computer, the web server not only sends the web browser the HTML file and associated CSS, image, audio, and video files in response to the original HTTP request it also sends a program associated with the webpage. When the user interacts with the webpage, the program, now on the user's computer, responds immediately. No information is sent

through the Internet, and in fact the user may disconnect from the Internet and the program will continue to function.

In general, client-side processing is strongly preferred when possible. It gives much faster response time, since program inputs and results do not need to be sent through the Internet. It also reduces load on web servers, since the programs are run on the web client's computer, not the web server. However, many applications require the program to run on the web server, because they need access to data on the server or more importantly the ability to modify data on the server. These include the login application and shopping application which you'll implement in your upcoming homework assignment. More advanced webpages can combine client-side processing with server-side processing.

The HTTP Protocol

The HTTP protocol is a critical component to the web. It provides a basic request response mechanism. The web client program (e.g. a web browser) sends a request for a specific resource, the web server sends a response. HTTP can be used to transfer files of any type. In fact the format of the files transferred is not part of the specification – the HTML language has its own much more complex specification. Let's take a closer look at HTTP.

Request

A request consists of a request line specifying the type of request and the URL of the resource being requested, followed by a series of headers, followed by the body of the request (which is often empty).

HTTP 1.1 (the current specification) supports a number of different request types. The most important are GET and POST, which are both requests for the web server to send back an actual file (typically a webpage, but potentially images, video, etc.) We will go over the difference between GET and POST in a section below. Other requests include HEAD, which acts the same as a GET except that the associated file isn't actually sent – it can be used to get meta-information about a file, like when it was last modified, without the overhead of having the actual file sent; DELETE as its name implies can be used to delete a resource; and OPTIONS gets information about a server or a resource. There are a few other request types in addition to these—check the HTTP 1.1 specification for the full list. By far the most common requests are GET and POST.

Following the request type and URL are a series of header fields. Header fields can be used for a variety of purposes including providing information on what types of files our web client supports, telling the web server that we are only interested in receiving the file if it has been modified after a specific date, or telling the web server that we want a specific character encoding used.

Response

A response consists of a status line, followed by a series of headers, followed by the actual file sent as the body of the response.

The status line provides the familiar HTTP error codes. 200 indicates that the request was successful; 404 says that the resource requested has not been found; 403 means that the resource exists, but we don't have permission to access it; 500 indicates a server problem; and so forth.

Header information for a response includes information on the type of data being sent and the size of the information sent. It can also provide information on how recently the data has been modified, or how long the information will be good for and thus how long the web browser can keep it in the browser cache.

GET and POST

One important distinction is between the GET and POST request types. The GET type is the standard request used for a link between one webpage to another. GET or POST can be used when the user enters information into a form and clicks on the submit button.

The most important difference between GET and POST is that GET is supposed to be idempotent – this means that multiple calls to GET should not get different results. This essentially means that GET should not change the state of the webserver. Consider the following examples:

- The user submits a message to an online bulletin board. This must be sent using POST, as it changes the state of the webserver.
- The user adds an item to a shopping cart. Again, this changes the state of the webserver and must be done using POST.
- The user looks up the phone number of a faculty member. This does not change the state of the webserver, so it may be done using GET or POST.

When the user enters information into a form, if the form submission uses GET than the form contents will be sent as part of the URL. Consider for example this simple form:

If we use GET the URL for the new webpage will look something like this:

```
http://www.example.edu/Login?name=Molly&password=FloPup
```

Notice that the name and password from the form is encoded within the URL. In contrast, if we use POST, the URL will look like this:

```
http://www.example.edu/Login
```

In a POST, the data entered in the form will be sent in the body of the request, not in the URL.

Each approach has advantages and disadvantages. For some situations, such as names and passwords, you won't want the data entered visible in the URL. However, placing the data in the URL allows the user to create a bookmark which includes the data entered within a form. A bookmark of a URL for a webpage generated using POST does not include the form data, and thus will not create a full record of what the user saw, when the page was previously visited.

MIME Types

One of the most important pieces of information sent using HTTP headers is the MIME type. MIME stands for Multipurpose Internet Mail Extensions. A MIME type specifies a type and a subtype. Here are some examples:

```
text/html
text/plain
text/xml
image/gif
image/jpeg
image/png
audio/mp3
video/mpeg
```

An HTTP response header includes a content-type field specifying the MIME type of the file contained within the response body. HTTP requests can provide an accept field specifying which types of media the web browser is willing to accept.