Lecture 2

Web server

Questions + test = 20 min

What Does it Mean to Host a Website?

Hosting a website means that you put your website files on a special computer called a **server**. This computer makes your website files publicly accessible on the internet, so anyone can visit it.

In order for other people around the world to see the website, these special computers have a particular set of software installed. This software is called a **web server.**

A web server's basic job is to receive incoming **requests** and **respond** by sending the requested page to the user's browse.

All websites on the internet use a web hosting company to host their websites. Even tech giants like Netflix (hosted on Amazon) and PayPal (hosted by Google Cloud hosting) use third-party service providers for their hosting.

There are only a handful of companies like Google, Microsoft, and Amazon that completely host all their services on their own platforms.

They can do this because they have the technical and financial resources to take on such a huge task. All other businesses (including us) use a web hosting company to host their website.

What are web servers used for?

Web servers are primarily used to process and manage HTTP/HTTPS requests and responses from the client system.

A web server can also perform several other functions, such as:

Store and protect website data: A web server can store and protect critical website data from unauthorized users.
Control bandwidth to regulate network traffic: A web server can help eliminate the downtime caused by high web traffic. Web hosts can set bandwidth to manage the rate of data transmission over the internet and minimize the excess network traffic.
Server-side web scripting: The server-side web scripting feature enables users to create dynamic web pages using scripting languages such as Ruby, Python, and PHP.
Virtual hosting: Web servers can also be used as virtual servers to run multiple applications, websites, data, and other services.

How web servers work

The end user processes a request via a web browser installed on a web server. The communication between a web server or browser and the end user takes place using Hypertext Transfer Protocol (HTTP). The primary role of a web server is to store, process, and deliver requested information or webpages to end users. It uses:

Physical Storage: All website data is stored on a physical web server to ensure its safety. When an end user enters the URL of your website or searches it using a keyword on a browser, a request is generated and sent to the web server to process the data.

Web browser: The role of web browsers such as Firefox, Chrome, or Internet Explorer is to find the web server on which your website data is located. Once the browser finds your server, it reads the request and processes the information.

Web server

The term *web server* can refer to hardware or software, or both of them working together.

On the hardware side, a web server is a computer that stores web server software and a website's component files

- HTML documents, images,
- CSS stylesheets,
- JavaScript files.

The server delivers these files to the client’s device. It is connected to the internet and can be accessed through a URL such as https://www.google.com/.

A web server connects to the Internet and supports physical data interchange with other devices connected to the web.

On the software side, a web server includes several parts that control how web users access hosted files. At a minimum, this is an *HTTP server*.

An HTTP server is software that understands URLs (web addresses) and HTTP (**Hypertext Transfer Protocol -** the protocol your browser uses to view webpages).

An HTTP server can be accessed through the **domain names** of the websites it stores, and it delivers the content of these hosted websites to the end user's device.

A server with just the HTTP server component is referred to as a *static* server, as opposed to a *dynamic* server which has several additional components.

A domain name - is a string of text that maps to an alphanumeric IP address, used to access a website from client software. In plain English, a domain name is the text that a user types into a browser window to reach a particular website. For instance, the domain name for Google is 'google.com'.

Who manages domain names?

Domain names are all managed by domain registries, which delegate the reservation of domain names to registrars (organizations that manage top-level domains (TLDs) '.com' and '.net' — specifically by maintaining the records of which individual domains belong to which people and organizations).

Anyone who wants to create a website can register a domain name with a registrar, and there are currently over 300 million registered domain names.

What is the difference between a domain name and a URL?

A uniform resource locator (URL), sometimes called a web address, contains the domain name of a site as well as other information, including the protocol and the path.

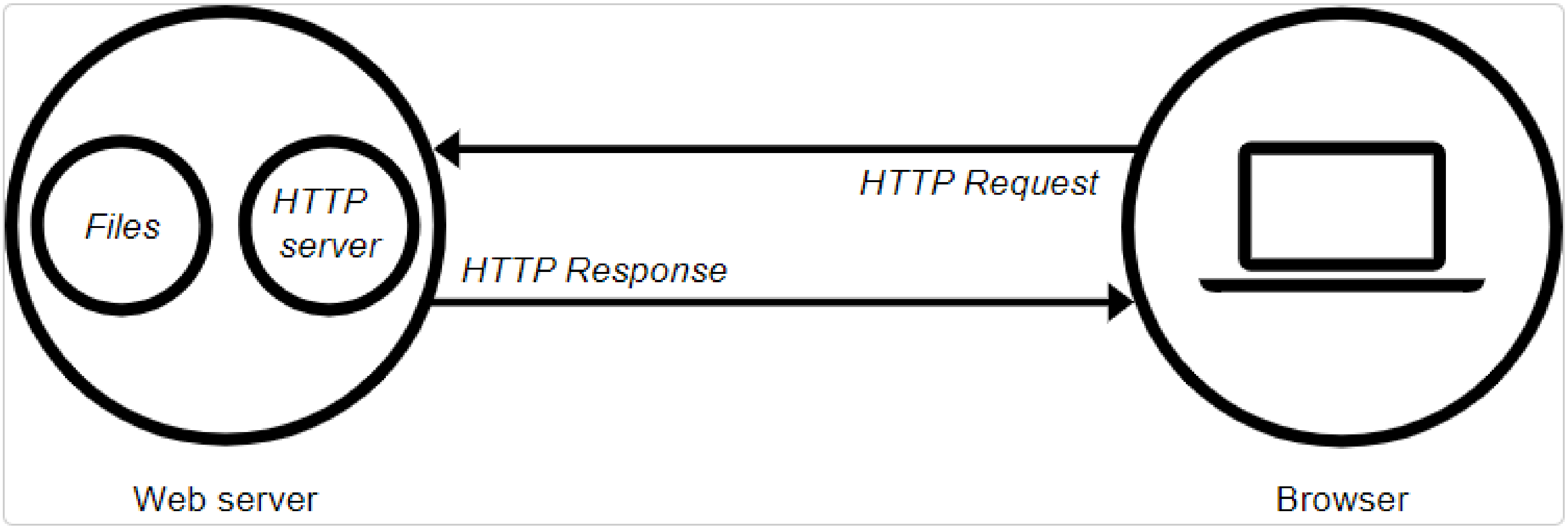
For example, in the URL 'https://cloudflare.com/learning/', 'cloudflare.com' is the domain name, while 'https' is the protocol and '/learning/' is the path to a specific page on the website.

At the most basic level, whenever a browser needs a file that is hosted on a web server, the browser requests the file via HTTP.

When the request reaches the correct (hardware) web server, the (software) *HTTP server* accepts the request, finds the requested document, and sends it back to the browser, also through HTTP.

(If the server doesn't find the requested document, it returns a 404 response instead.)

Client-server communication through HTTP shown on the picture



Static vs. dynamic servers

To publish a website, you need either a static or a dynamic web server.

A **static web server**, or stack, consists of a computer (hardware) with an HTTP server (software).
It is "static" because the *server sends its hosted files as-is to your browser.*

A **dynamic web server** consists of a static web server + extra software, most commonly an *application server* and a *database*.

It is "dynamic" because the application server updates the hosted files before sending content to your browser via the HTTP server.

For example, to produce the final webpages you see in the browser, the application server might fill an HTML template with content from a database.

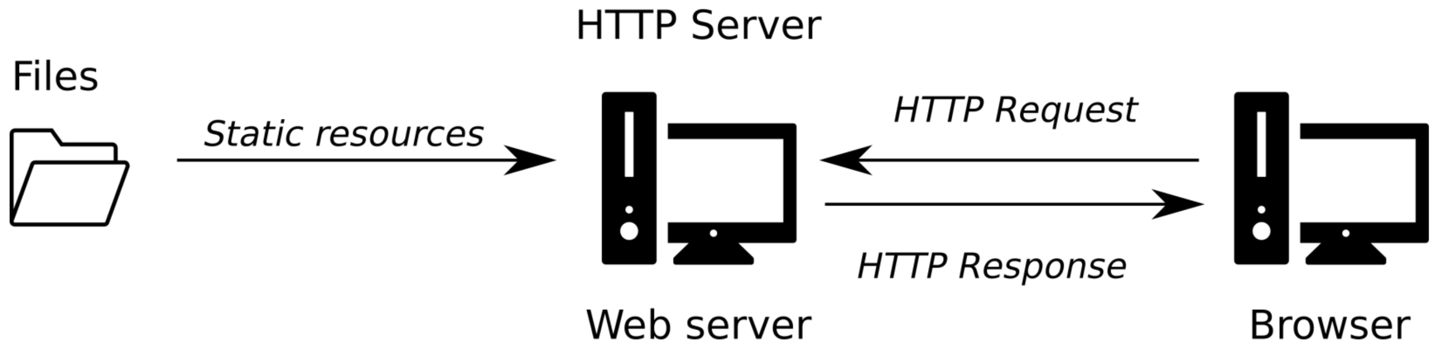
Static servers

A **static server** consists of a computer (hardware) with just an HTTP server (software). The static server can only respond to GET requests for pre-existing HTML documents, or other types of files, and send those documents to the browser. While loading the HTML document, the browser may send further GET requests for other pre-existing files linked in the HTML code, such as CSS, JavaScript, images, and so on.

For example, suppose we are vising a hypothetical website focused on travel locations, and we are navigating to a specific page on travelling to France, at
http://www.travel.com/locations/france.html.

In case the website is served using a static server, there is an actual france.html document on the server. All the server has to do **is send you a copy of that file** (as it is shown on the picture)

As another example, the online version of online book —is also hosted on a static server. This means that all of the HTML documents comprising the website are prepared in advance. Entering a URL for a specific page (such as web-servers.html) sends the appropriate file to your browser through HTTP.



Dynamic servers

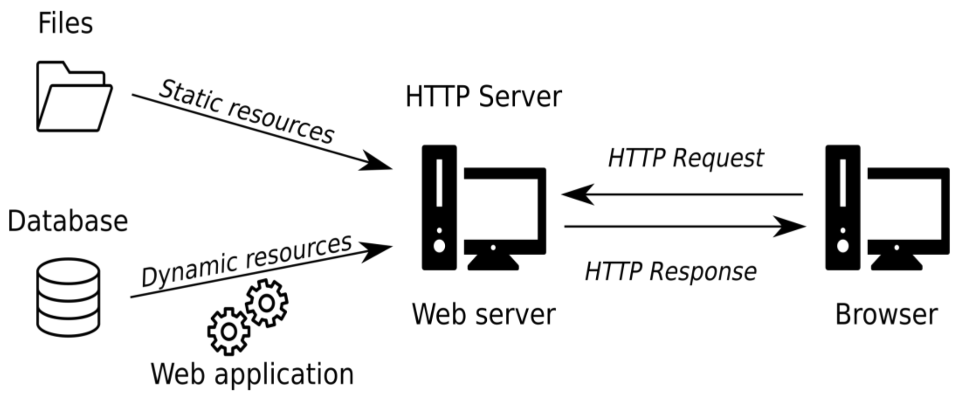
A **dynamic server** consists of an HTTP server plus extra software, most commonly an application server and a database. It is "dynamic" because the server dynamically builds the HTML documents, or any other type of content, before sending them to your browser via the HTTP server.

Typically, the dynamic server uses its application server, i.e., software running server-side scripts, HTML templates, and a database, to assemble the HTML code. Once assembled, the dynamically assembled HTML content is sent via HTTP, just like static content.

With a dynamic server, when entering the above-mentioned hypothetical URL, http://www.travel.com/locations/france.html, into the address bar in our browser, the france.html document *doesn't exist yet*.

The server waits for your request, and when the request comes in, it uses various "elements" (e.g., templates, a database, etc.) and a "recipe" to create that page on the spot, just for you.

Sites like Wikipedia have thousands of webpages. Typically, these kinds of sites are composed of only a few HTML templates and a giant database, rather than thousands of static HTML documents. This setup makes it easier to maintain and deliver the content.



Communicating through HTTP Web protocols and HTTP.

HTTP is a protocol specifying the way that communication between the client and the server takes place.

As its name—Hypertext Transfer Protocol—implies, HTTP is mainly used to transfer hypertext (i.e., HTML documents) between two computers.

HTTP is not the only protocol in use for communication on the web. For example, **FTP** and **Web socket** are examples of other web communication protocols.

However, HTTP is the most basic and most commonly used protocol.

Almost everything we do online is accomplished through HTTP communication.

The secured version of HTTP, known as **HTTPS**, is becoming a very common alternative to HTTP and thus should be mentioned.

However, HTTPS just adds a layer of security through encrypted communication and is not fundamentally different from HTTP.

In the context of the web, a protocol is a set of rules for communication between two computers.

HTTP, specifically, is a textual and stateless protocol:

- **Textual** means that all commands are plain-text, therefore human-readable.
- **Stateless** means that neither the server nor the client remember previous communications.

For example, an HTTP server, relying on HTTP alone, cannot remember if you are "logged-in" with a password, or in what step you are in a purchase transaction.

Furthermore, **only clients can make HTTP requests**, and then only to servers. **Servers can only respond to a client's HTTP request**.

When requesting a file via HTTP, clients must provide the file's URL.
The web server must answer every HTTP request, at least with an error message.
For example, in case the requested file is not found the server may return the "404 Not Found" error message.
The 404 error is so common that many servers are configured to send a customized 404 error page.

HTTP methods.

The HTTP protocol defines several *methods*, or verbs, to indicate the desired action on the requested resource on the server.

The two most commonly used HTTP methods, for a request-response between client and server:

- GET
- POST

GET—Used to **request** data from the server
POST—Used to **submit** data to be processed on the server

There are a few other methods, such as PUT and DELETE, which are used much more rarely.

The GET method

The GET method is used to request data.
It is by far the most commonly used method in our usual interaction with the web.

For example, typing a URL in the address bar of the browser in fact instructs the browser to send a GET request to the respective server.
A static server is sufficient for processing GET requests in case the requested file is physically present on the server.

The response is usually an HTML document, which is then displayed in the browser.

In addition to manual typing in the browser address bar, GET requests can also be sent programmatically, by running code.

The POST method

The POST method is used when the client sends data to be processed on the server.

It is more rarely used compared to GET, and somewhat more complicated.

For example, there is no way to send a POST request by simply typing a URL in the browser address tab, unlike with GET. Instead, making a POST request to a web server can only be made through code, such as JavaScript code.

Also, a dynamic server is required to process POST requests, where server-side scripts determine what to do with the received data.

POST requests are preferred over GET requests when we need to send substantial amounts of data to the server.

Software

Running a static server is easy to do on our own. Assuming you already have our web page documents prepared.
An HTTP server is included in many software packages and libraries, and it does not require any special installation or configuration.
There are numerous software solutions which can start an HTTP server in a few minutes;

There are also several **free cloud-based** options to have a **managed static server** such as **GitHub Pages** , which means you do not even need to have your own dedicated computer or invest in paid cloud-based services to run your static server.

There are also professional HTTP server software packages, used for building both static and dynamic servers. Two most commonly used ones are **Apache HTTP Server** and **Nginx**.

Setting up and running a dynamic server is more complicated, requiring specialized installation, configuration, and maintenance. There are no instant solutions, such as the ones we will see shortly for static servers , since it is up to us to define the way in which the server dynamically generates HTML content.
It is done by setting up an application server and writing custom **server-side scripts**. With a dynamic server, in addition to the HTTP server software, you need to write server-side scripts which run on the server, as opposed to client-side JavaScript scripts that run on the client.

Server-side scripts are responsible for tasks such as generating customized HTML content, authentication, managing user sessions, etc. There are several programming languages (and frameworks) that are commonly used for writing server-side scripts, such as **PHP**, **Python (Django)**, **Ruby (on Rails)**, and **JavaScript (Node.js)**.

Web server software list

Some of the most common web servers are outlined below:

- **Linux web server software:** Linux server is built on an open-source Linux operating system that enables you to deliver content, applications, and services to end users. Linux servers are flexible, consistent, and high-performing servers with snapshot capabilities, optimized security, and scalable cloud technologies. These servers help address the increasing requirements of web services, applications, database management, and more.
- **NGINX web server software:** NGINX is a popular open-source web server that runs and utilizes resources efficiently. It can handle huge volumes of traffic. It offers reverse proxy, HTTP caching services, email proxy, and load balancing. NGINX is a scalable, lightweight, and powerful web server capable of handling concurrent connections and is ideal for delivering static content.
- **Apache web server software:** Apache web server or Apache HTTP server is an open-source server that processes user requests and delivers web assets and content via HTTP. This web server uses the MySQL database to store critical information in an easily readable format. With the help of the PHP programming language, Apache can create and serve dynamic web content.
- **IIS web server software:** Microsoft Internet Information Service (IIS) web server is also known as a Windows web server. It's one of the most commonly used web servers used on a Windows operating system. It is a versatile and stable web server widely used to host ASP.NET web applications, static websites, and web applications built on PHP. It can also be used as an FTP server to host WCF services. Although it has a built-in authentication option such as Windows, ASP.NET, and Basic, it's easier for Windows users to sign in to various web applications using their domain account. Other built-in security features include TLS certificate management, request logging, FTP-specific security options, and more.

Web server vs. application server differences

- **Web server:** The web server accepts and processes requests from end users for static website content. It handles requests and responses via HTTP only. Web servers are generally helpful in serving static content or static HTML webpages. It consumes fewer resources such as CPU or memory compared to the application server and provides a runtime environment for web applications.

- **Application server:** The application server can deliver web content and dynamic content required for displaying decision support, transaction results, or real-time analytics. However, its primary role is to enable interaction between the end user and server-side application code. These servers enhance interactive content or website components depending on the request. Application servers use web containers. These servers use more resources compared to web servers and provide the runtime environment for enterprise applications. These servers also support HTTP and RPC/RMI protocols.

Practical considerations

There are advantages and disadvantages to both the static and the dynamic server approaches.

Static sites (i.e., sites served with a static server) are simple, fast, and cheap, but they are harder to maintain (if they are complex) and impersonal.

Dynamic sites provide more flexibility and are easier to modify, but also slower, more expensive, and technically more difficult to build and handle.

We will only build static sites hosted using a static web server.
A static server cannot use a database or template to send personalized HTML content, just pre-compiled HTML documents. A static server is not limited to showing fixed, non-interactive content.

For example, the HTML content of the web page can be modified in response to user actions through client-side scripts (in JavaScript), without needing a server.

Later on, we will also see that static pages can dynamically "grab" information from other locations on the web, again using client-side JavaScript, including from existing dynamic servers and databases.

That way, we can integrate dynamic content even though we do not operate our own dynamic server. Still, there are things that can only be accomplished with a dynamic server.

The most notable example where a dynamic server is an obvious (and only) solution is **authentication**.

For example, suppose we want to create a password-protected website. In our website, we can add a form with an input element where the user enters a password, and only if the password is valid—the content will be shown.

This requires authentication—some way to evaluate the validity of the entered password.

Now, suppose we have a database of valid passwords for the various authorized website users. Where can we place that database, and how can our page access it?

If we place it directly on the client, e.g., as an array in our JavaScript code, the contents will be exposed to anyone looking at the source code of our page.

Remember that whenever a JavaScript script is linked to our HTML document, the user can access and view the code.

Even if we place the password database on a different location, such as a separate static server, we still need to hard-code the instructions for accessing that other location in our JavaScript code, so that the web page can access it.

Again, anyone who reads those instructions can access the database the same way the browser does. The solution is to send the user-entered password for validation using a server-side script. If the password is valid, the server can return an “OK” message and/or any content that the specific user is allowed to see.

That way, the password database is not accessible, since the server is not allowed to send it—only to accept an entered password and compare it to those in the database.

URLs and file structure

URLs and index.html

A static server is associated with a directory on the computer, serving its contents over the web.

Once the server is running, a client can request any HTML document (or other type of file) which is located inside that directory, or any of its sub-directories, by entering a **Uniform Resource Locator (URL)** in the browser address bar.

To construct the right URL for accessing a given HTML document (or other resource), one needs to know two things:

The **IP address** of the host computer and
the **port** where the server is running,

or, alternatively, the **domain name**.

Let's go over the separate components this URL is composed of:

http://
means we are communicating using **HTTP**. This part is automatically completed by the browser, so it can be omitted.

159.89.13.241

is the **IP address** of the web server that hosts the website. The IP address is a unique identifier given to a computer connected to a network, making it possible to identify the computer in the network.

:8000

is the **port** number where the server is running.
When using the default port for a given communication protocol, which is 80 for HTTP and 443 for HTTPS, the port number can be omitted.

/web-mapping/web-servers.html
is the location of the **document**.

With a static server, this means that within the directory we are serving there is a sub-directory named web-mapping, and inside it there is an HTML document named web-servers.html.

If you delete the last name of the file you still should see the index.html page even though it wasn't not specify any HTML file name.

This happens because standard protocol dictates that a file named index.html will be provided by default when we navigate to a directory, rather than an HTML document, on the web server.

The index.html file usually contains the first page users see when navigating to a website.

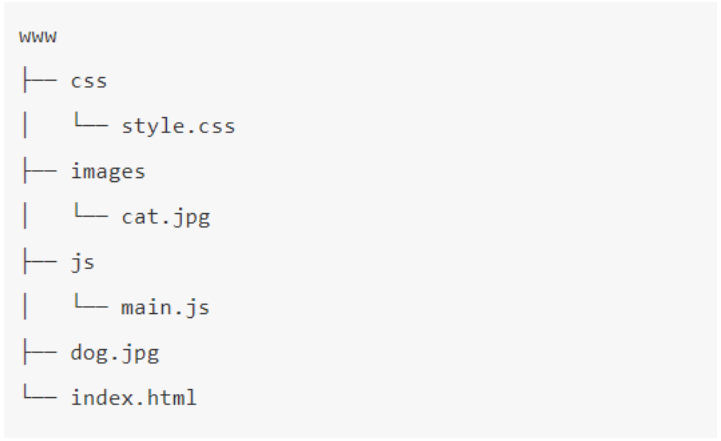
Usually navigation goes to a *textual* URL such as https://www.google.com, rather than a *numeric* IP address and port number, such as http://216.58.198.68:80/.

A **Domain Name Server (DNS)** - uses its resources to resolve the domain name into the IP address for the appropriate web server.

This saves us the trouble of remembering IP addresses, using more recognizable textual addresses instead.

File structure

The following diagram shows a hypothetical file structure of a static website directory:



```
www
├── css
│   └── style.css
├── images
│   └── cat.jpg
├── js
│   └── main.js
├── dog.jpg
└── index.html
```

The various files that comprise the website are located either in the **root directory** or in **sub-directories** within the root directory.

In the above example, www represents the root directory of the website.

In this example, the root directory www contains a default index.html document.

This means that when we browse to the directory address without specifying a file name, the index.html file is sent by default.

The root directory www also contains sub-directories.

Here, the sub-directories are used for storing additional files linked to the hypothetical HTML code of index.html:

- css—for CSS files (.css)
- images—for images
- js—for JavaScript files (.js)

The structure and names of the sub-directories are entirely up to us, web page developers.

We can even place all of the files in the root directory, without any internal division to sub-directories.

However, it is usually convenient to have the type of sub-directory structure as shown above, where files of different types are stored in separate sub-directories.

That way, the various components that make up our website can be easier to track and maintain.

Relative paths

In the above file structure example, the images folder contains an image file named cat.jpg.

In case we want this image to be displayed on the index.html page, the HTML code in index.html needs to include an <img> element. The src attribute of that <img> element needs to refer to the cat.jpg file. Either one of the following versions will work:

<img src="/images/cat.jpg"**>** **<img src**="images/cat.jpg"**>**

Both versions of the src attribute value are known as *relative* file paths, because they are relative to a given location on the server:

In the first case, the path is relative to the **root** directory of the server, specified by the initial / symbol.

In the second case, the path is relative to the **current** directory where the HTML is loaded from, so the path just starts with a directory or file name in the same location (without the / symbol).

In this particular example the index.html file is in the root directory, thus the current directory is identical to the root directory.

Therefore the cat.jpg file can be reached with either /images/cat.jpg or images/cat.jpg. Incidentally, we have another image file dog.jpg in the root directory (which, again, is the current directory for index.html).

CSS and JavaScript

CSS and JavaScript code can be loaded from separate files, usually ending in .css and .js, respectively.

This approach is preferred to using embedded CSS or JavaScript code.

Keeping CSS and JavaScript code in separate files takes a little more effort than embedding it directly in the HTML document, but saves work as our sites become more complex, because:

1. Instead of repeating the same CSS and JavaScript code in different pages of the website, we can load the same file in all pages.
2. When modifying our external CSS or JavaScript code, all web pages loading those files are immediately affected.

Linking CSS

Linking an external CSS file can be done using the <link> element within the <head> of the HTML document.

For example, in our hypothetical static server file structure the **css** folder contains a **style.css file.**

This CSS file can be linked to the index.html document be including the following <link> element:

<link rel="stylesheet" **href**="/css/style.css"**>**

In this case, it makes sense to use a ***relative path*** (a relative file path points to a file relative to the current page*)* which is relative to the root directory, since a website usually has a single set of CSS and JavaScript files.

That way, exactly the same <link> element can be embedded in all HTML documents, regardless of where those HTML documents are placed.

Linking JavaScript

Linking a JavaScript code file can be done by adding a file path in the **src** attribute of a **<script>** element.

The <script> element can then be placed in the <head> or the <body> of the HTML document.

For example, our hypothetical file structure has a folder named js with a JavaScript file named main.js.

That script can be loaded in the ***index.html*** document by including the following <script> element:

<script src="/js/main.js"**></script>**

Again, a relative path, relative to the root directory, is being used.

Running a static server

So far we have discussed several background topics related to running a static server:

- Communication through HTTP
- Difference between static and dynamic servers
- Components of a URL and the file structure on the server

What is left to be done is actually *running* a server, to see how it all works in practice.

We will experiment with running a static web server using two different methods:

- A **local server**, using your own computer and Python
- A **remote server**, using the GitHub Pages platform

Local with Python

Setup instructions

The local option for running a static server.

This example will demonstrate the HTTP server built into Python, which only requires you have **Python** installed (you can check if Python is installed by opening the **Command Prompt** (open the **Start** menu, then type cmd) and typing python.

If you see a message with the Python version number, then Python is installed and you have just entered its command line, marked by the >>> symbol.

You can exit the Python command line by typing **exit()** and pressing **Enter**.

In case you see an error message, such as the following one, then Python is not installed:

```
'python' is not recognized as an internal or external
command, operable program or batch file.
```

Running the server

To run Python's HTTP Server:

Open the **Start** menu and type **cmd** to enter the Command Prompt.

Navigate to the directory that you want to serve (e.g., where your index.html file is), using **cd** (change directory) followed by directory name.

*For example, if your directory is in drive D:\, inside a directory named Data and then a sub-directory named server, you need to type **cd D:\Data\server** to navigate to that directory.*
*Type the expression python **-m http.server.***

You should see a message such as the following , meaning that the server is running:

```
Serving HTTP on 0.0.0.0 port 8000 …
```

As evident from the above message, the default port where the Python server is running is 8000.

In case you want to use a different port, such as 4000, you can specify the port number as an additional parameter:

```
python -m http.server 4000
```

To stop the server, press **Ctrl+C**.

Testing served page

Once the server is running, you can access the served web page(s) by navigating to the following address in a web browser, assuming there is an **index.html** file in the root of the served directory:

```
http://localhost:8000/
```

In case you want to load an HTML document other than **index.html**, or if the document is located in one of the sub-directories of the server, you can specify the path to the HTML file you wish to load.

For example:

```
http://localhost:8000/locations/france.html
```

If you initiated the server on a different port, replace the 8000 part with the port number you chose, such as 4000:

```
http://localhost:4000/locations/france.html
```

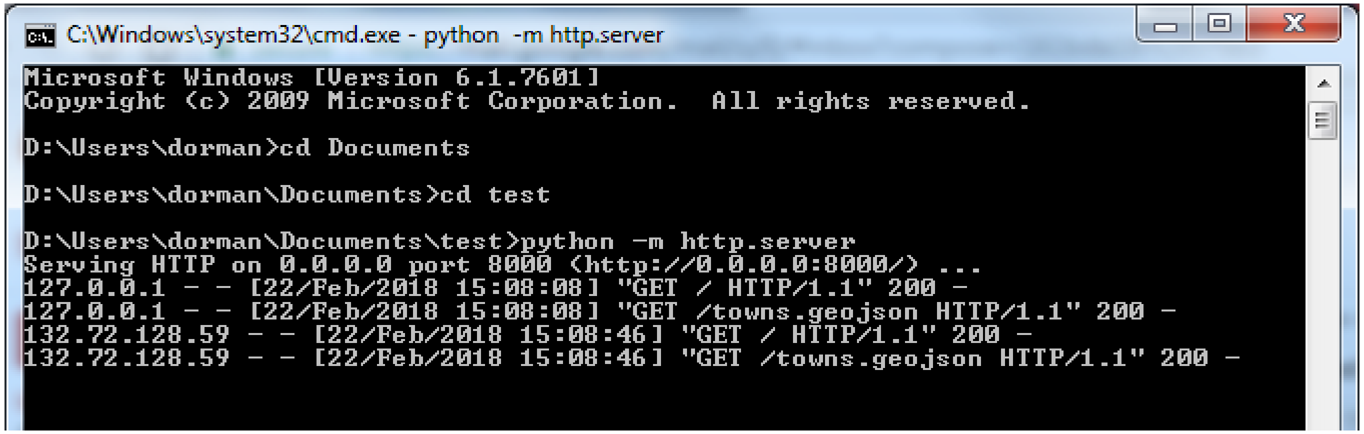
The word localhost means you are accessing *this computer*.

The server and the client are the same computer. It is extremely useful for development and testing of websites.

Having the client and server on the same computer means that we can simulate and test the way that a client accesses the website, without needing to deploy the website on a *remote* server which requires some more work.

Python's HTTP server prints incoming requests in the console, which is useful for monitoring how the server works.

For example, the screenshot in Figure shows several logged messages printed while using the server.
In this particular printout, we can see that the server successfully processed four GET requests, all of which took place on February 22, in 2018, at around 15:08



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

D:\Users\dorman>cd Documents

D:\Users\dorman\Documents>cd test

D:\Users\dorman\Documents\test>python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [22/Feb/2018 15:08:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [22/Feb/2018 15:08:08] "GET /towns.geojson HTTP/1.1" 200 -
132.72.128.59 - - [22/Feb/2018 15:08:46] "GET / HTTP/1.1" 200 -
132.72.128.59 - - [22/Feb/2018 15:08:46] "GET /towns.geojson HTTP/1.1" 200 -
```

Remote with GitHub Pages

Python’s HTTP server is simple enough to start working with, but there are other difficulties if you intend to use it for **production**, i.e., in real-life scenarios, where stability is essential.

For instance, you need to take care of the network administration issues such as making sure your server has an IP address that can be reached from other computers, i.e., a public IP address, and that the server is not behind a firewall.

In addition, you need to make sure the IP address of the computer always stays the same (a static IP address), take care of the hardware of your server, such as making sure the computer is always running and connected to the internet, make sure that the server is restarted in case the computer restarts, and so on.

Using a remote hosting service, we basically let other people handle all of that. In other words, we don’t need to worry for any of the hardware, software, and network connection issues—just the contents of our website.

There are numerous hosting services for static web pages.

For example, both **Google** and **Amazon**, as well as many other smaller companies, offer *paid* static hosting services.

We will use the **GitHub** platform for hosting our static web page hosting, which is *free*.

Although GitHub is mainly a platform for online storage of **Git** repositories and collaborative code development, one of its “side” functions is that of a static server.

The static server functionality of GitHub is known as **GitHub Pages**.
Using GitHub Pages as a remote static server has several advantages for our purposes:

- It is **simple**.
- It is **free**.
- It is part of **GitHub**, a popular platform for collaborative code development, which is useful to become familiar with.

Another good alternative is **surge.sh**. It is also free, and can be quicker to set up compared to GitHub Pages, but requires using the command line.

Git and GitHub

When working with code, it becomes important to keep track of different versions of your projects.

This allows you to undo changes made weeks or months ago.

Versioning becomes even more important when collaborating with others, since in that case you may need to split your project into several “branches”, or “merge” the changes contributed by several collaborators back together.

To do all of those things, people use **version-control systems**.

One of the most popular revision control systems around today is **Git.**

GitHub

Git is a version-control system, Git projects are also called repositories.

GitHub is a web-based Git repository hosting service. Basically, GitHub is an online service where you can store your Git repositories, either publicly or privately.

The platform also contains facilities for interacting with other people, such as raising and discussing issues or subscribing to updates on repositories and developers you are interested in, creating a community of online code-collaboration.

For anyone who wants to take part of open-source software development, using Git and GitHub is probably the most important skill after knowing how to write the code itself.

For any public GitHub repository, the user can trigger the **GitHub Pages** utility to serve the contents of the repository.

As a result, the contents of the repository will be automatically hosted at the following address:
https://GITHUB_USER_NAME.github.io/REPOSITORY_NAME/

where:
- GITHUB_USER_NAME is the **user name**
- REPOSITORY_NAME is the **repository name**

Everything about static servers applies in remote hosting too. The only difference is that the served directory is stored on another, remote server, rather than your own computer.

For example, in order for a web page to be loaded when one enters a repository URL as shown above, you need to have an index.html file in the root directory of your GitHub repository

Setup instructions

Instructions for running a remote static server on GitHub Pages.

To host our website on GitHub pages, go through the following steps:

1. Create a GitHub account on https://github.com/ (in case you don't have one already), then sign-in to your account. Your username will be included in all of the URLs for GitHub pages you create, as in GITHUB_USER_NAME in the URL shown above.

2. Once you are logged-in on https://github.com/, click the + symbol on the top-right corner and select **New repository**
3. Choose a name for your repository. This is the REPOSITORY_NAME part that users type when navigating to your site, as shown in the above URL.

Make sure the *Initialize this repository with a README* box is checked. This will create an (empty) README.md file in your repository, thus exposing the **Upload files** screen which we will use to upload files into our repository.

4. Click the **Create repository** button The newly created repository should be empty, except for one file named README.md

5. Click on the **Settings** tab to reach the repository settings.
Inside the settings page, select the **Pages** tab from the menu on the left, to see the **GitHub Pages** settings section.

6. In the **Source** panel, instead of **None** select **main** and click the **Save** button Go back to the repository page and click the **Add files** button, then the **Upload files** button from the dropdown menu. This will take you to the file upload screen .

7. Drag and drop all files and folders that comprise your website into the box. This should usually include at least an HTML document named **index.html.**

8. Wait for the files to be transferred. Once all files are uploaded, click the **Commit changes** button

10. Your website should be live at http://GITHUB_USER_NAME.github.io/REPOSITORY_NAME/. Replace GITHUB_USER_NAME and REPOSITORY_NAME with your own GitHub user name and repository name, respectively·

Types of Website Hosting Services

Not all websites are the same. Some are smaller and have low traffic, while others are larger with more content and visitors.

A smaller website requires fewer resources, such as disk space and bandwidth. On the other hand, a larger and more popular website will need more resources to run efficiently.

Website hosting companies provide different types of hosting offers designed to address specific website needs.

The most common hosting options are :

Shared Hosting

A shared hosting service is suitable for small websites, blogs, and small businesses that are just starting out.

They are able to keep their costs down by allowing multiple websites to share the same server resources. This makes hosting your website affordable.

Suitable for: Starting a new blog or small business website.

Shared hosting provider: Bluehost

VPS Hosting

VPS hosting (Virtual Private Server hosting) is still a shared hosting environment. It offers a flexible set of resources to handle large traffic spikes.

You get a partitioned-off private server for your website that you can manage from your hosting control panel. This gives you the best of both worlds, the low cost of shared hosting with the flexibility of dedicated resources.

Suitable for: Medium-sized businesses, popular blogs, and eCommerce stores.

VPS hosting company: HostGator

Managed WordPress Hosting

Managed WordPress hosting is a specialized hosting service made specifically for WordPress.

On a managed hosting platform, the hosting company takes care of updates, backups, and caching of your website.

This allows you to focus on creating content and growing your business.

Suitable for: Popular blogs, business websites, membership websites.

Dedicated Hosting

A dedicated server hosting gives you the entire server dedicated to your own website.

You get all the resources of the server, advanced tools for server management, the ability to install your own software, and even your own operating system.

You'll be managing your own server, which may require some technical skills. It is an advanced option for larger websites that need high-performance to tackle higher traffic volume.

Suitable for: Enterprise-level businesses, hugely popular websites, eCommerce stores.

Dedicated hosting company: SiteGround or HostGator.

Questions

1. What is a web server? Kinds of Servers.
2. Can you explain the difference between an application server and a web server?
3. What is HTTP is?
4. Explain the Difference between URL and domain name
5. Web server software.

Test

1. There is an arrangement where you allow an agency to host your website for you on their Web Server for a fee, this is called as ?

Web Service

Web Hosting

Web Marketing

None of these

2. The World Wide Web is a massive collection of web sites, all hosted on

computers

Internet

World Wide Web

a network

3. The web server (computer) where your web site's html files, graphics, etc. reside is known as the

web localhost

web host

web server

None of these

4. WWW is the acronym of the

Web World Wide

World Wide Webpages

World Wide Web

World Wide Websites

The Web is a computer network all over the

world wide

country wide

continent wide

state wide

6. All the computers in the Web can communicate or transfer the data(text file) and graphics (picture file) with each other.

False

True

Not always

None of these

7. A Web browser access the web page from a web server by a

request

response

Interrupts

normal messages

8. A request is a standard HTTP request containing a

computer address

page address

domain address

MAC address

9. Which service is used to resolve the domain names to IP addresses ?

Resolver

Name-Converter

DNS

None of these

10. What is an ISP ?

Internet Service Provider

Internet State Provider

Intranet Service Provider

Internet State Provider

11. Which of these is not an expenses in the Web Service ?

Hardware Expenses

Software Expenses

Labor Expenses

Transprot Expenses

12. A domain name is the unique text name corresponding to the _______ of a computer on the Internet.

numeric MAC address

numeric network address

numeric IP address

All the above

13. What is used as a separator in a domain name ?

dot(.)

slash(/)

colon(:)

All the above

14. Users to your web site will often connect via a

switch

hub

modem

None of these

Thank you !