

Web System Development

Module 00: Introduction

Why the Web?

People go to websites for many different reasons, though most of these fall into a few broad categories:



Information

- [Wikipedia](#)
- [BBC News](#)
- [Khan Academy](#)
- [WebMD](#)
- [Stack Overflow](#)



Entertainment

- [YouTube](#)
- [Netflix](#)
- [Twitch](#)
- [Spotify](#)
- [Steam](#)



Social

- [Facebook](#)
- [LinkedIn](#)
- [Discord](#)
- [Reddit](#)
- [WhatsApp Web](#)



Creative Expression

- [Instagram](#)
- [DeviantArt](#)
- [SoundCloud](#)
- [Medium](#)
- [Behance](#)

Different visions for the web

Simple document viewer

- Static HTML pages
- Hyperlinks and text documents
- Read-only content
- Examples:
 - [Wikipedia](#)
 - [BBC News](#)
 - [NASA](#)
 - [MDN Web Docs](#)
 - [arXiv](#)
 - [Project Gutenberg](#)

Powerful application platform

- Rich UIs and interactivity
- Client-side logic with JavaScript
- Web APIs, SPAs, real-time data
- Examples:
 - [Spotify](#)
 - [Piskel](#)
 - [Figma](#)
 - [Google Docs](#)
 - [YouTube](#)
 - [Photopea](#)
 - [Ableton Learningsynths](#)
 - [0hh1](#)
 - [Classic Minecraft](#)
 - [Soundtrap](#)

Different ways of delivering content



Static Website

- Always returns the same hard-coded content (pre-built HTML pages)
- Built with HTML, CSS, and maybe some JavaScript
- Each page is a separate file on the server
- No server-side logic or database
- Same content for everyone
- Fast, secure, and easy to host
- **Examples:** Documentation, blogs, portfolios



Dynamic Website

- Content is generated on the fly **per user** (server-side or client-side)
- Personalized content and user interactions
- Uses databases, APIs, user auth, etc.
- Can scale and change content dynamically
- **Examples:** Gmail, Facebook, E-commerce platforms

Real-World Examples



Static

- [Bootstrap Docs](#)
- [Gatsby Sites](#)
- [Personal portfolios](#)



Dynamic

- [Gmail](#)
- [Amazon](#)
- [Airbnb](#)

When to Use Each?

- Use **Static** for fast-loading, simple sites (docs, blogs)
- Use **Dynamic** for interactive, personalized, data-driven apps

What's the Web?

- What do you think of when you hear the terms *the web* or *the Internet* ?
- When most people refer to *the web* , they are talking about the World Wide Web (aka *WWW*).
- Although *the web* and *the Internet* are commonly used as though they mean the same thing, the web is actually part of the Internet.
- You can think of the web as the **websites and content** you access when you are online using a browser.
- You can think of the Internet as the **roads** that connect computers and other devices. It is along these **roads** that content is delivered to your device.

How does the Internet work?

- The internet is a **global network** connecting computers and devices for information sharing, enabling activities like browsing websites, sending emails, and streaming videos.
- It acts as a vast *web linking everything* , facilitating communication and access to online resources and services worldwide.
- The Internet is the backbone of the Web, the technical infrastructure that makes the Web possible.

The Internet (1960s-1970s)

- The United States Department of Defense wanted an information system that could survive an *attack* . They needed a system that could keep working even if a part or parts of the system were *destroyed* . So, they designed and created a *network of devices* connected in a web-like design with no central location. Information could be sent from one device to any other device or devices along different paths.
- This network was called **ARPANET** (Advanced Research Projects Agency Network). It was the first network to use a method of breaking down data into packets, and sending them separately, possibly along different paths, and reassembling them at their destination. This method is known as *packet switching* . This method of sending data is still used today.
- Elizabeth Fienler was an American information scientist who directed the Network Information Center (NIC) for the ARPANET from 1972 to 1989. She and her team created and maintained the ARPANET Resource Handbook, the directory of people and services, the protocol handbook, and the Request for Comments (RFCs).
- In 1974, Vint Cerf and Bob Kahn introduced **TCP/IP** - Transmission Control Protocol (TCP) and Internet Protocol (IP). These protocols allow multiple networks to connect and communicate with each other. This is the foundation of today's Internet.
- [Wikipedia](#)

Birth of the World Wide Web (1990s)

- The infrastructure of the *Internet* needed a user-friendly interface.
- In 1989, British scientist Tim Berners-Lee proposed a system that made the Internet accessible to everyone, the *World Wide Web* .
- In 1990, Berners-Lee developed the first web *browser* and *web server* .
- The first website, info.cern.ch , went live in 1991.
- This was the beginning of the public's access to the World Wide Web.

The Explosion of the Web (Late 1990s-2000s)

- With the expansion of the World Wide Web, people needed a way to navigate, search its contents, and also present and share information with others.
- **Web browsers** were created to allow people access to websites.
- **Search engines** were created to index the information on the World Wide Web and make this information easier to find.
- **Social media** sites and **e-commerce** sites were created to help people connect, communicate, and conduct business with each other.
- **Netscape Navigator** and **Internet Explorer** were two of the earliest web browsers.
- **Yahoo!** and **AltaVista** were two early search engines.
- **Facebook** and **Amazon** were created to connect people and allow e-commerce.

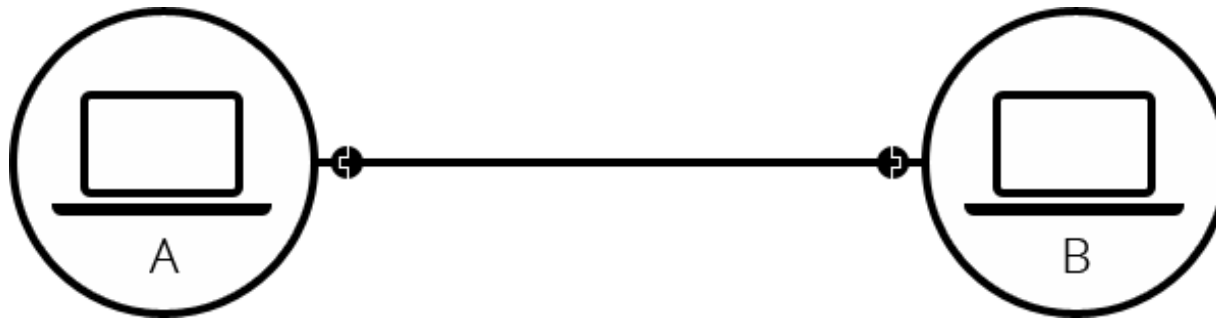
What is the Internet?

What is the Internet?



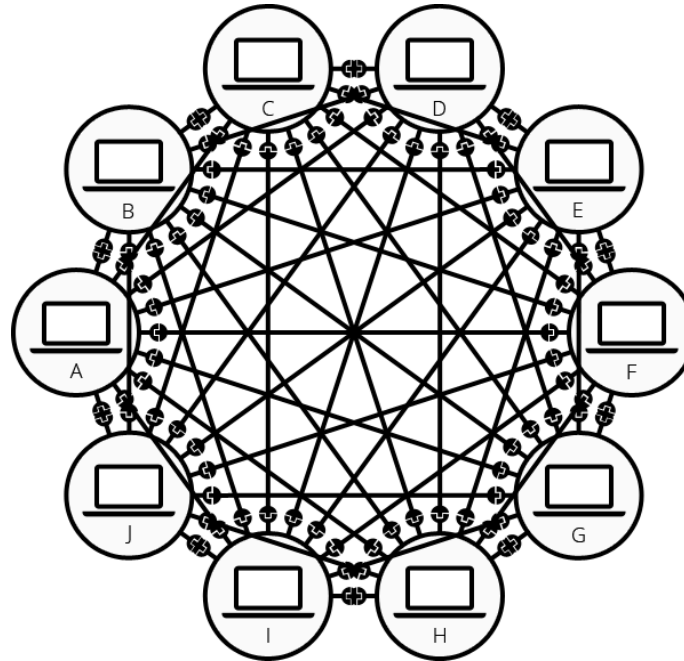
A simple network

When two computers need to communicate, you have to link them, either physically (usually with an Ethernet cable) or wirelessly (for example with Wi-Fi or Bluetooth systems). All modern computers can sustain any of those connections.



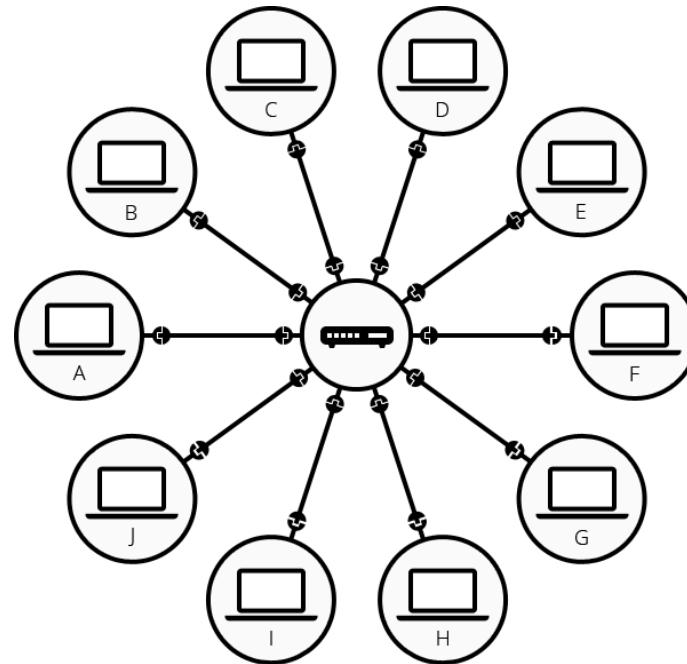
Expanding the network

- Such a network is not limited to two computers.
- You can connect as many computers as you wish.
- But it gets complicated quickly. If you're trying to connect, say, ten computers, you need 45 cables, with nine plugs per computer!



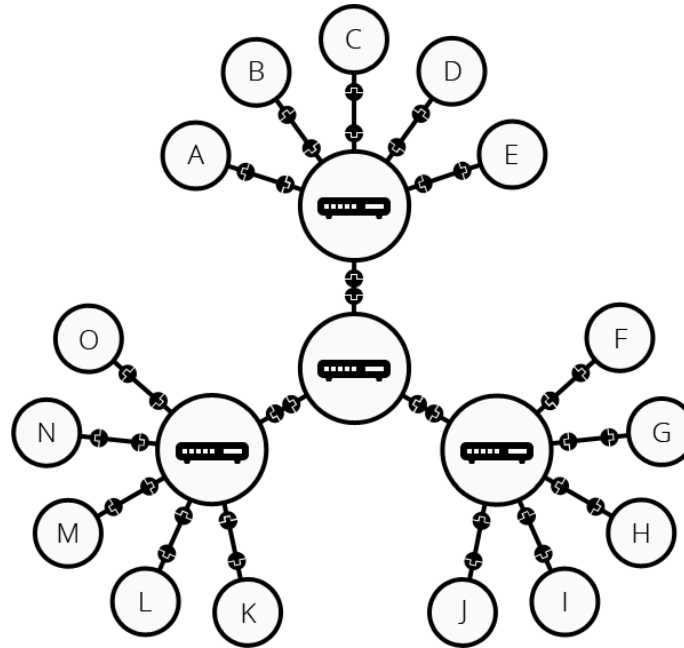
A real network

- To solve this problem, each computer on a network is connected to a special tiny computer called a network **switch** (or switch for short).
- This switch has only one job: like a signaler at a railway station, it makes sure that messages sent from a given computer arrive only at their target destination computer.
- Once we add a switch to the system, our network of 10 computers only requires 10 cables: a single plug for each computer and a switch with 10 plugs.



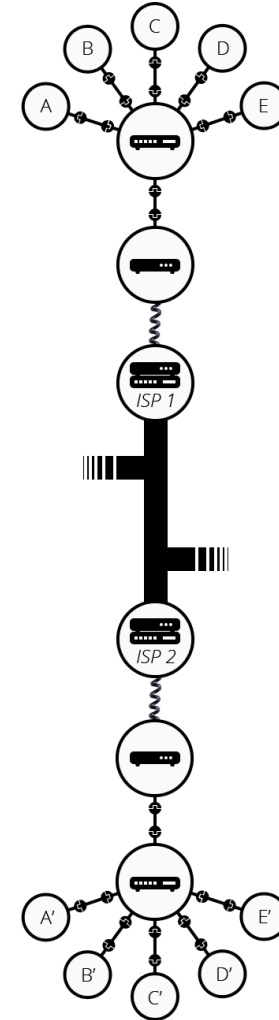
A network of networks

- So far so good. But what about connecting hundreds, thousands, billions of computers?
- Of course a single switch can't scale that far, but, if you read carefully, we said that a switch is a computer like any other, so what keeps us from connecting two switches together? Nothing, so let's do that.



Internet Service Providers (ISPs)

- The next step is to send the messages from our network to the network we want to reach.
- To do that, we will connect our network to an Internet Service Provider (ISP).
- An ISP is a company that manages some special routers that are all linked together and can also access other ISPs' routers. So the message from our network is carried through the network of ISP networks to the destination network.
- The Internet consists of this whole infrastructure of networks.



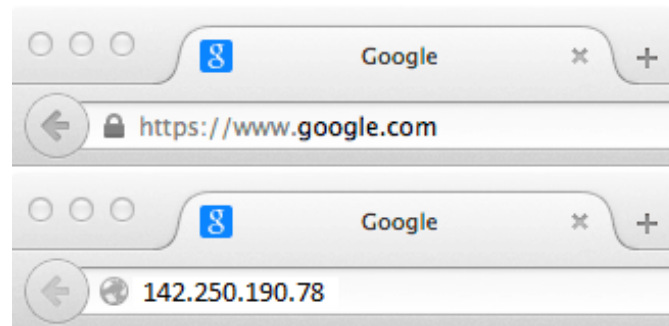
Packets, Routing, and Reliability

The Internet: Packets, Routing & Reliability



Finding computers

- If you want to send a message to a computer, you have to specify which one.
- Thus any computer linked to a network has a unique address that identifies it, called an *IP address* (where IP stands for Internet Protocol).
- It's an address made of a series of four numbers separated by dots, for example:
`192.0.2.172`
- That's perfectly fine for computers, but we human beings have a hard time remembering that sort of address.
- To make things easier, we can alias an IP address with a human-readable name called a *domain name* . That's where DNS comes in.



IP Addresses and DNS

The Internet: IP Addresses & DNS



The hidden network that makes the internet possible

The hidden network that makes the internet possible - Saja...



Submarine Cable Map

<https://www.submarinecablemap.com>

How the Web Brings you Websites

How the Web Brings You Websites



What is a 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** . 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: a software that understands **URLs** (web addresses) and HTTP.
 - Hypertext Transfer Protocol (HTTP) is the protocol used to transfer data over the web.
 - 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.

Uniform Resource Locator (URL)

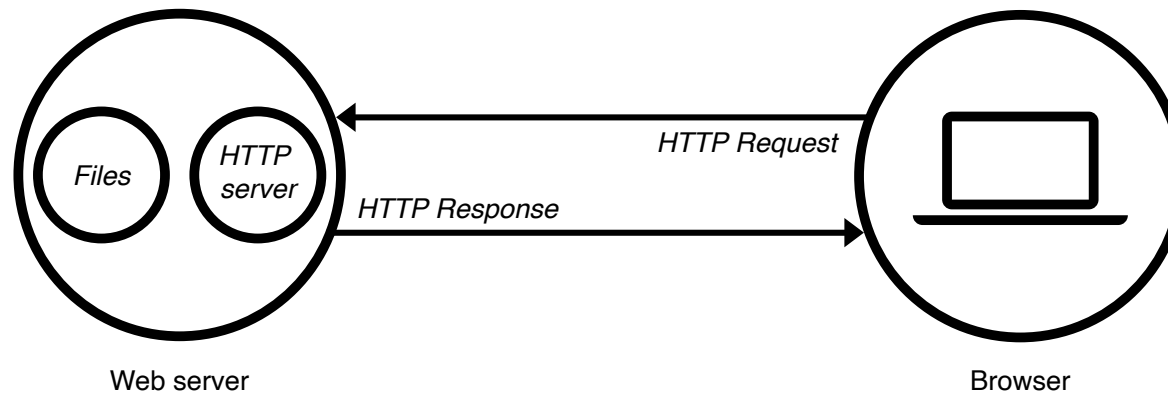
A URL is a string of characters that specifies where a resource can be found on the web.

`https:// example.com : 81 /a/b/c ? user=Alice&year=2025 #fragment`

- `https://` is the scheme
- `example.com` is the host
- `81` is the port
- `/a/b/c` is the path
- `?` is just a separator
- `user=Alice&year=2025` is the query
- `fragment` is the fragment

Requesting a file via HTTP

- 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.)

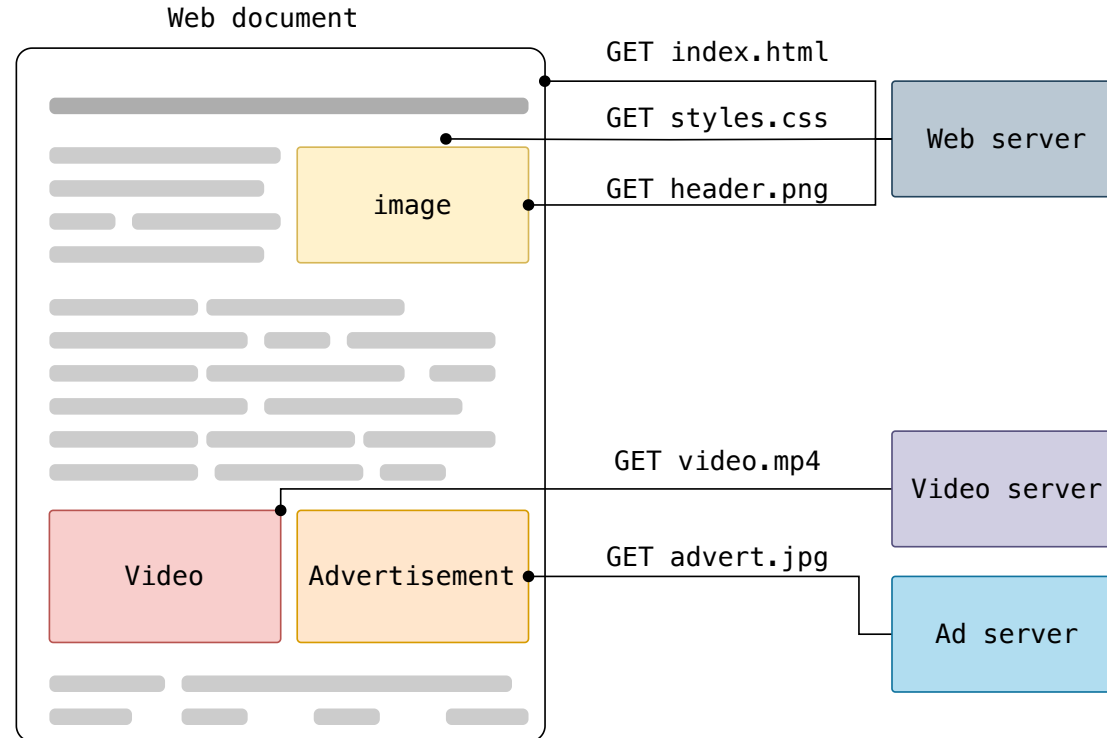


HTTP and HTML

The Internet: HTTP & HTML



Overview of HTTP



- HTTP is a **stateless** protocol, meaning that the server does not remember anything about the client.
- It's a **request-response** protocol, meaning that the client sends a request to the server and the server sends a response back.

HTTP Messages

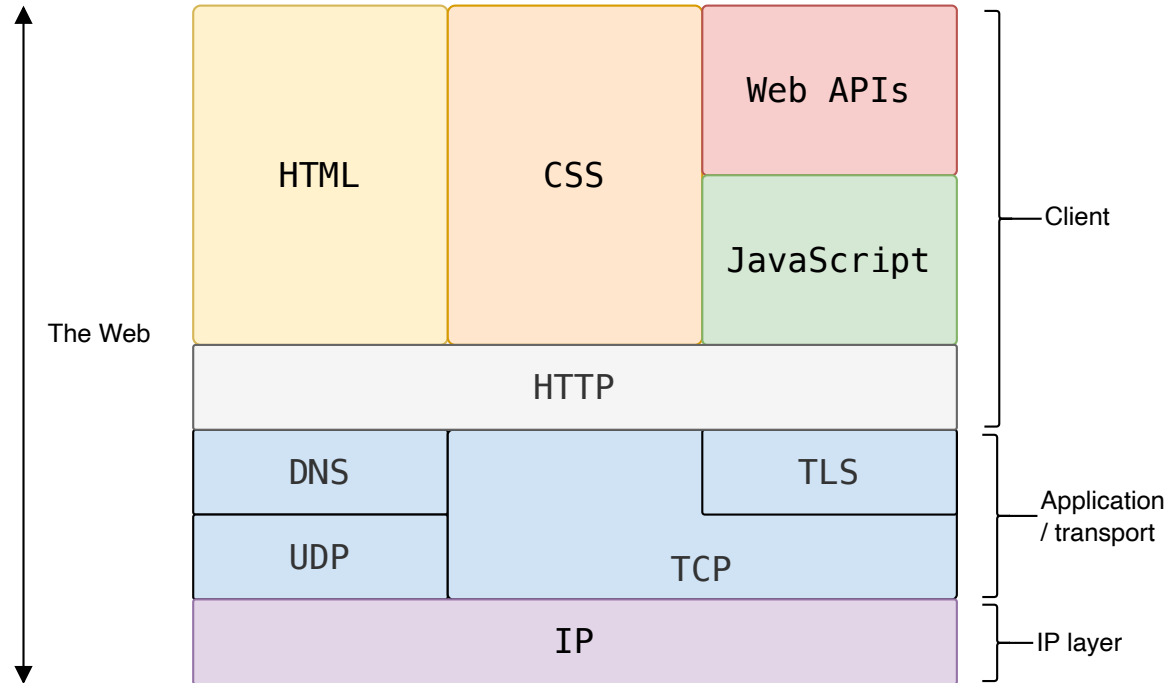
Request

```
GET / HTTP/1.1  
Host: developer.mozilla.org  
Accept-Language: fr
```

Response

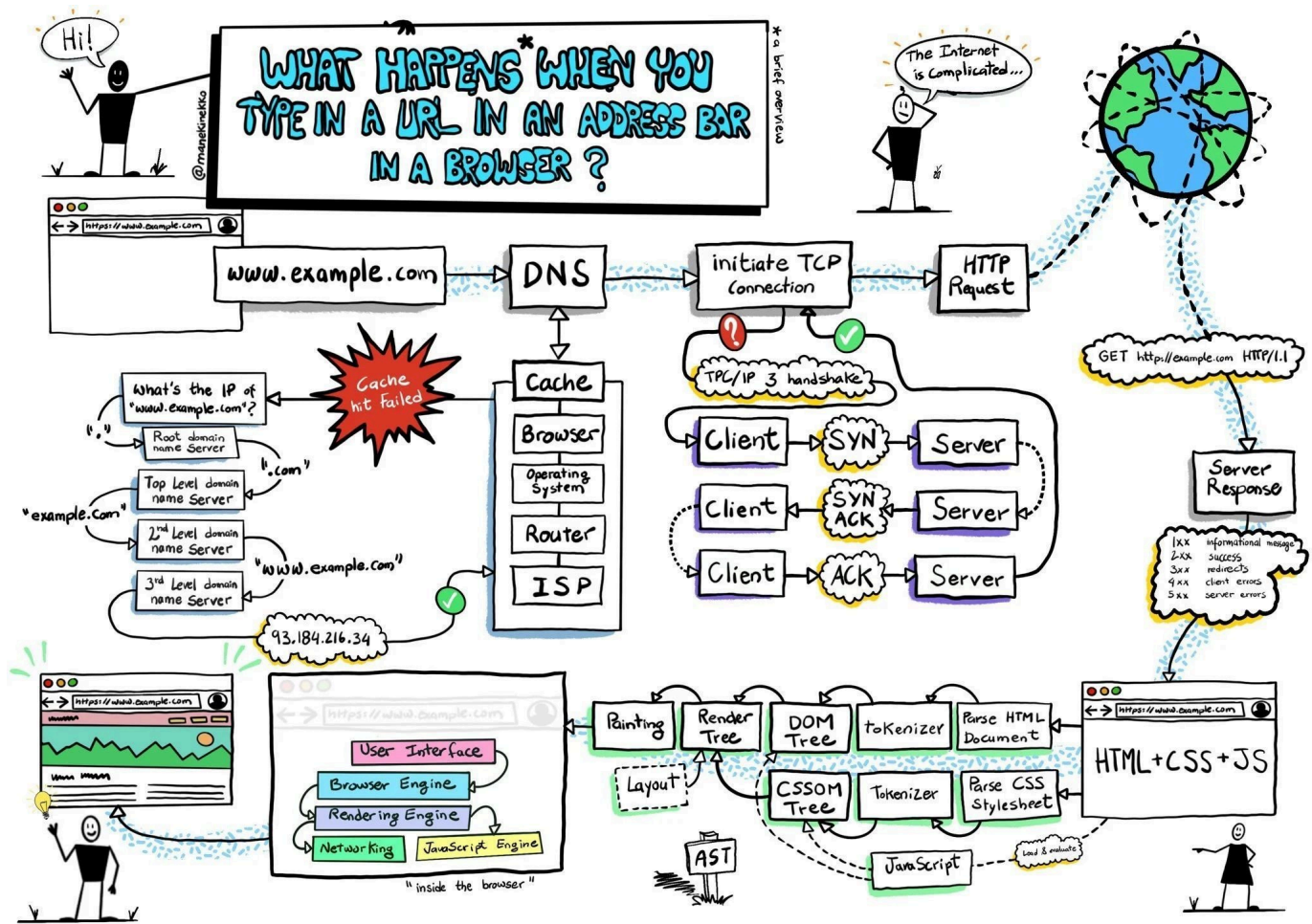
```
HTTP/1.1 200 OK  
Date: Sat, 09 Oct 2010 14:28:02 GMT  
Server: Apache  
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT  
ETag: "51142bc1-7449-479b075b2891b"  
Accept-Ranges: bytes  
Content-Length: 29769  
Content-Type: text/html  
  
<!doctype html>... (here come the 29769 bytes of the requested web page)
```

Stack of protocols



- HTTP is an application layer protocol that is sent over **TCP** , or over a **TLS-encrypted** TCP connection, though any reliable transport protocol could theoretically be used.
- Due to its extensibility, it is used to not only fetch hypertext documents, but also images and videos or to post content to servers, like with HTML form results.
- HTTP can also be used to fetch parts of documents to update Web pages on demand.

**What happens when you type a URL
in your browser?**



Layers of Software

It doesn't matter if you're building a website, a mobile app, a game, or an embedded system... You'll usually find the same layers across all systems:

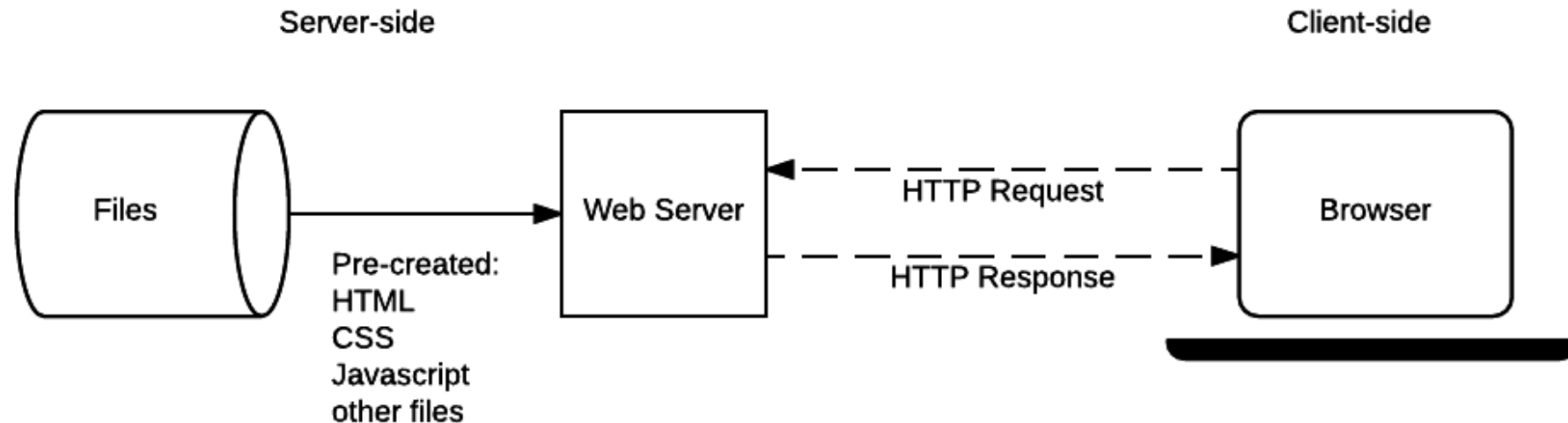
- **User Interface** (The presentation layer) — the point of interaction between the user and the system. It can be a web page, a mobile app, a desktop application, or even a console interface.
- **API** (Application Programming Interface) — Exposes the application's logic and defines how other modules or services interact with it. This includes REST, GraphQL, gRPC, SOAP, or WebSockets, but also local interfaces such as shared libraries, drivers, or SDKs.
- **Business Logic** (The logic layer) — Contains the business rules and core functionality. It can be implemented in different languages and organized into additional layers (services, controllers, use cases). This is the layer that gives meaning to the application.
- **Database** (The data layer) — Manages data persistence. It's not limited to relational databases — it can also include in-memory storage, files, queueing systems, or even physical sensors in embedded environments.
- **Infrastructure** (The execution layer) — This could be a cloud server, a Docker/Kubernetes cluster, an on-premise mainframe, a mobile device, or specialized hardware. This layer defines the scalability, resilience, and availability of the software.

Client Server Request Response

- Web browsers communicate with web servers using the Hyper Text Transfer Protocol (HTTP).
- When you click a link on a web page, submit a form, or run a search, an **HTTP request** is sent from your browser to the target server.
- The request includes a *URL* identifying the affected resource, a *method* that defines the required action (for example to get, delete, or post the resource), and may include additional information encoded in *URL parameters* (the field-value pairs sent via a query string), as POST data (data sent by the HTTP POST method), or in associated cookies .
- Web servers wait for client request messages, process them when they arrive, and reply to the web browser with an **HTTP response** message. The response contains a status line indicating whether or not the request succeeded (e.g., "HTTP/1.1 200 OK" for success).
- The body of a successful response to a request would contain the requested resource (e.g., a new HTML page, or an image), which could then be displayed by the web browser.

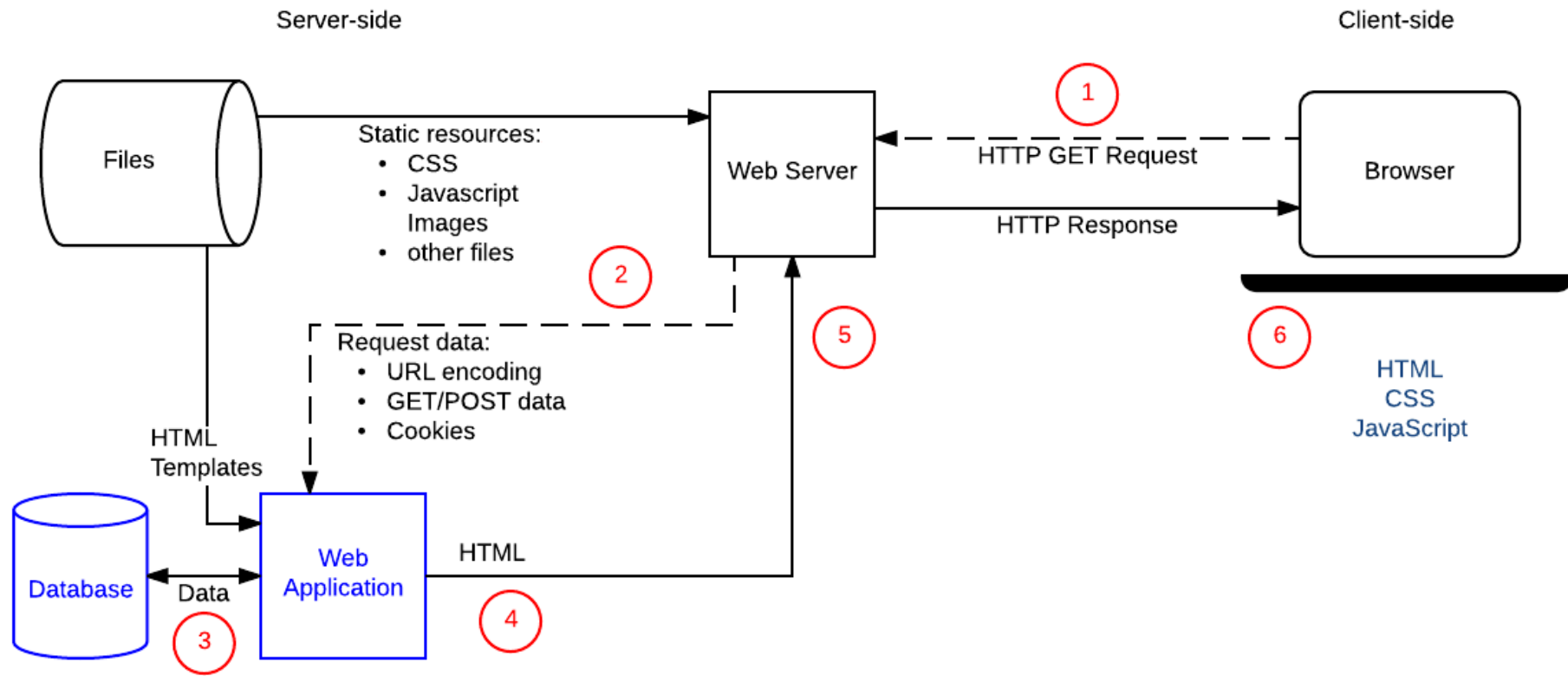
Static sites

A static site is one that returns the same hard-coded content from the server whenever a particular resource is requested). When a user wants to navigate to a page, the browser sends an HTTP "GET" request specifying its URL.



Server-side programming

- On a dynamic website, HTML pages are normally created by inserting data from a database into placeholders in HTML templates (this is a much more efficient way of storing large amounts of content than using static websites).
- A dynamic site can return different data for a URL based on information provided by the user or stored preferences and can perform other operations as part of returning a response (e.g., sending notifications).
- Most of the code to support a dynamic website must run on the server. Creating this code is known as **server-side programming** (or sometimes **back-end scripting**).



Client-side programming

- **Client-side programming** (also called **front-end**) refers to code that runs in the user's web browser.
- It controls the behavior and appearance of web pages that users interact with directly
- **Key technologies:** HTML, CSS, and JavaScript
- **Examples of functionality:**
 - Interactive forms and user input validation
 - Dynamic content updates without page reloads
 - Animations and visual effects
 - Responsive design and layout adjustments
 - Local data storage and manipulation
- **Advantages:** Fast response times, reduced server load, better user experience with immediate feedback
- **Limitations:** Cannot access server resources directly, depends on user's browser capabilities, security restrictions

Are server-side and client-side programming the same?

Let's now turn our attention to the code involved in server-side and client-side programming. In each case, the code is significantly different:

- They have different purposes and concerns.
- They generally don't use the same programming languages (the exception being JavaScript, which can be used on the server- and client-side).
- They run inside different operating system environments.

Code running in the browser is known as **client-side code** and is primarily concerned with improving the appearance and behavior of a rendered web page. This includes selecting and styling UI components, creating layouts, navigation, form validation, etc.

By contrast, **server-side** website programming mostly involves choosing *which content* is returned to the browser in response to requests. The server-side code handles tasks like validating submitted data and requests, using databases to store and retrieve data and sending the correct data to the client as required.

Types of web developers

- The **frontend** is the stuff you see on the website in your browser, including the presentation of content and user interface (UI) elements. Front-end developers use *HTML* , *CSS* , *JavaScript* , and their relevant frameworks to ensure that content is presented effectively and that users have an excellent experience.
- The **backend** refers to the guts of the application, which live on the server. The back end stores and serves program data to ensure that the front end has what it needs. This process can become very complicated when a website has millions of users. Back-end developers use programming languages like *Java* , *Python* , *Ruby* , and *JavaScript* to work with data.
- **Full-stack developers** are comfortable working with both the front and back ends.

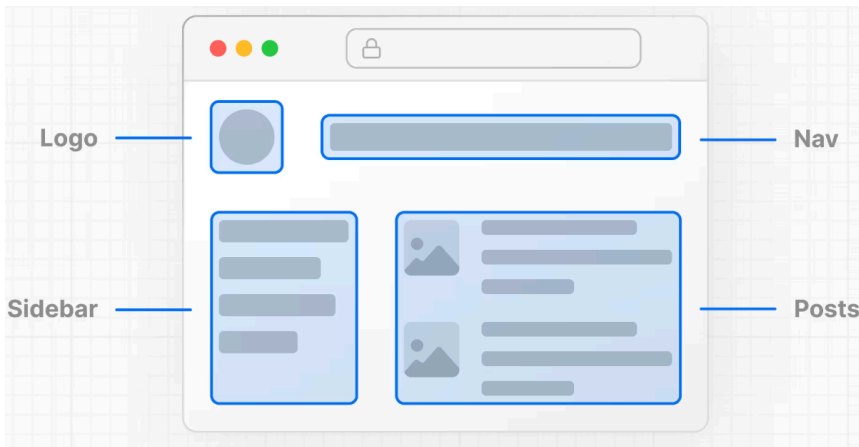
Frontend vs Backend

Frontend






- User interface (UI)
- HTML, CSS, JavaScript
- Runs in the browser
- Interacts with backend via APIs

Backend

- Business logic & data storage
- Authentication & authorization
- Node.js, Express, databases
- Runs on the server
- Sends data to the frontend



Why Split Frontend & Backend?

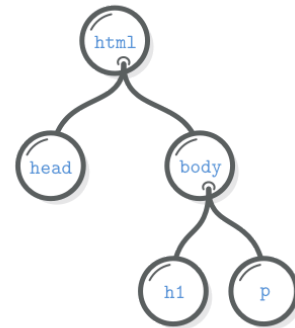
-  Easier to scale and maintain large projects
-  Test frontend and backend independently
-  Backend can serve multiple frontends (web, mobile)
-  Keep sensitive logic (auth, database) on the server
-  Enables API-based architecture (REST, GraphQL)

HTML, CSS, & JavaScript

HyperText Markup Language (**HTML**), Cascading Style Sheets (**CSS**), and **JavaScript** are the languages that run the web.

- HTML is for adding meaning to raw content by marking it up.
- CSS is for formatting that marked up content.
- JavaScript is for making that content and formatting interactive.

Think of **HTML** as the abstract text and images behind a web page, **CSS** as the page that actually gets displayed, and **JavaScript** as the behaviors that can manipulate both HTML and CSS.



HTML



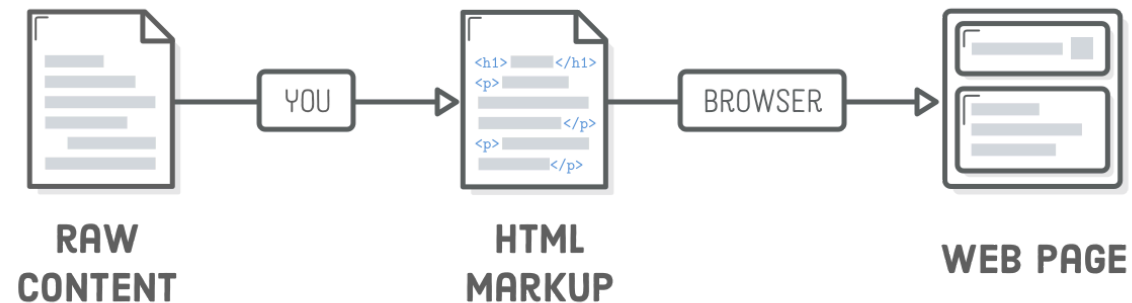
CSS



JAVASCRIPT

Basic Web Pages

HTML defines the content of every web page on the Internet. By “marking up” your raw content with HTML tags, you're able to tell web browsers how you want different parts of your content to be displayed.



For example, you might mark some particular run of text as a paragraph with this **HTML** :

```
<p id="some-paragraph">This is a paragraph.</p>
```

Then, you can set the size and color of that paragraph with some **CSS** :

```
p {  
  font-size: 20px;  
  color: blue;  
}
```

And, if you want to get fancy, you can re-write that paragraph when the user clicks it with some **JavaScript**

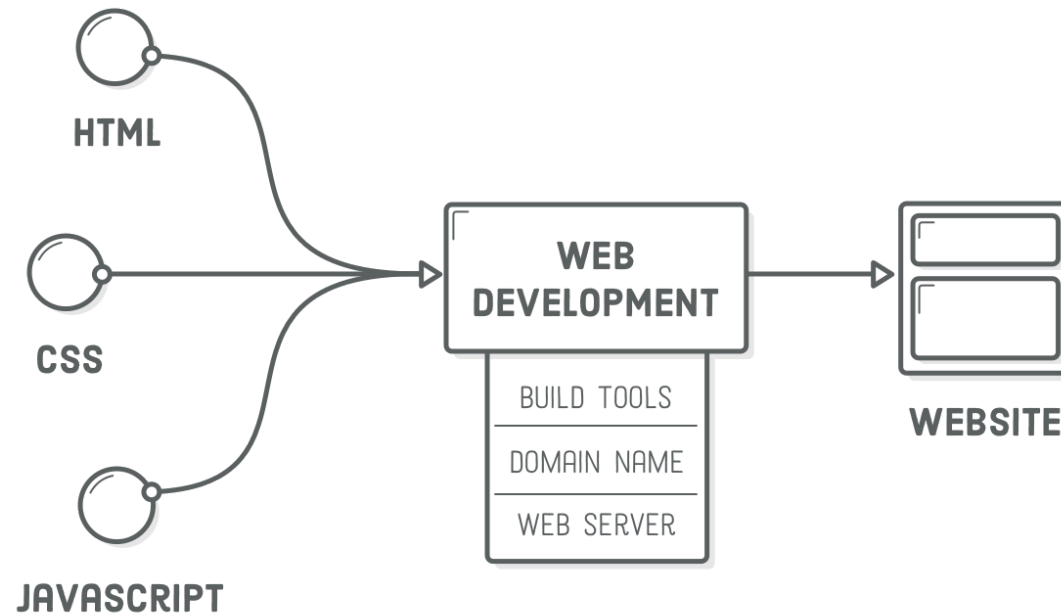
```
var p = document.getElementById('some-paragraph');  
p.addEventListener('click', function(event) {  
  p.innerHTML = 'You clicked it!';  
});
```

As you can see, HTML, CSS, and JavaScript are totally different languages, but they all refer to one another in some way. Most websites rely on all three, but the appearance of every website is determined by HTML and CSS.

Web Development

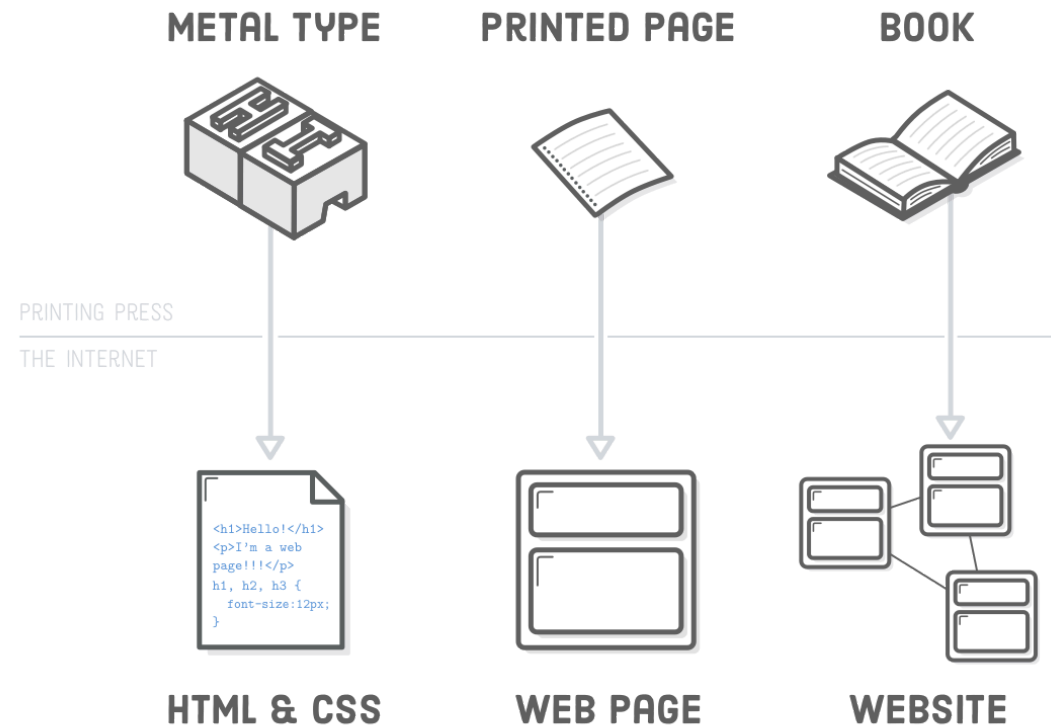
Unfortunately, mastering HTML, CSS, and JavaScript is only a prerequisite. There are a bunch of other practical skills that you need to run a website:

- Organizing HTML into reusable templates
- Standing up a web server
- Moving files from your local computer to your web server
- Reverting to a previous version when you screw something up
- Pointing a domain name at your server



Web Publishing

- Learning HTML is similar to the *printing industry* .
- Back in the days of the original printing press, printers created documents by arranging metal characters, dipping them in ink, and pressing them onto a piece of paper.
- In a lot of ways, that's exactly what web developers do, except instead of arranging moveable type, they write HTML and CSS.



Fundamentals, not Frameworks

- There's all sorts of front-end web development frameworks out there ([Bootstrap](#) , [ZURB foundation](#) , and [Pure CSS](#) , just to name a few).
- The goal of every single one of them is to abstract away some of the redundant aspects of creating web pages from scratch.
- These kinds of frameworks are an important part of real-world web development, and they're definitely worth exploring—but only after mastering the basics.
- This is stuff that stays with you forever, in spite of shiny new additions to the **HTML and CSS standards** or trendy new frameworks that help you do things faster.

Development Environment

- Code Editor
- Web Browser
- Node.js & Package Manager
- Local HTTP Server
- Version Control
- Database
- Docker

Code Editor

- We'll use **Visual Studio Code (VS Code)**
- Free, open source, and widely adopted
- Includes syntax highlighting, IntelliSense, Git integration
- Supports extensions like Prettier, ESLint, and Docker
- Download: code.visualstudio.com

Web Browser

- We'll primarily use Google Chrome
- Modern web platform with strong developer tools
- Key feature: **DevTools** (Elements, Console, Network, Performance)
- Other good options: Firefox Developer Edition, Safari, Edge

Node.js

- JavaScript runtime built on Chrome's V8 engine
- Runs JavaScript code on the server
- Foundation for our backend development and build tools
- Comes with **npm** (Node Package Manager)

Why Node.js?

- 🧠 Same language (JavaScript) for frontend and backend
- ⚡ Fast execution with non-blocking I/O (event-driven architecture)
- 🌐 Massive ecosystem with **npm** (largest package registry)
- 📦 Ideal for APIs, microservices, and full-stack development

Package Manager

- **npm** comes with Node.js — it's the default
- Allows installing and managing third-party libraries
- We'll use it to install tools like:
 - React, Express, Tailwind
 - Testing libraries
 - Build tools
- Commands: `npm install` , `npm run` , `npx`

Installing Node.js

1. Go to nodejs.org
2. Download the **LTS version** (recommended for most users)
3. Run the installer and follow the steps
4. After installation, check the version with `node -v` and `npm -v`
5. Create a new file called `hello.js` , write the following code and run it with `node hello.js` :

```
const message = `Hello World!`;
console.log(message);
```


How to setup a local HTTP server?







- Normally you can just open the HTML file in your browser directly.
- If the web address path starts with `file://` followed by the path to the file on your local hard drive, a local file is being used.
- In contrast, if the web address will start with `http://` or `https://` , to show that the file has been received via HTTP.

The problem with testing local files

Some examples won't run if you open them as local files. This can be due to a variety of reasons, the most likely being:

- They feature **asynchronous requests**: Some browsers (including Chrome) will not run async requests (see [Learn: Making network requests with JavaScript](#)) if you just run the example from a local file. This is because of security restrictions (for more on web security, read [Website security](#)).
- They feature a **server-side language**: Server-side languages (such as PHP or Python) require a special server to interpret the code and deliver the results.
- They **include other files**: Browsers commonly treat requests to load resources using the `file://` schema as cross-origin requests. So if you load a local file that includes other local files, this may trigger a **CORS** error.

What is Vite?

-  Vite is a fast build tool and development server for modern web projects
-  Lightning-fast dev server with hot module reload
-  Optimized production build with Rollup under the hood
-  Out-of-the-box support for modern JavaScript features
-  Works well with React, TypeScript, Tailwind CSS, and more
-  Simple to set up and extend

Installing Vite

1. Open your terminal
2. Run the following command:

```
npm create vite@latest
```

3. Choose:
 - Project name
 - Framework (e.g. React)
 - Language (e.g. TypeScript)

4. Navigate into your project:

```
cd your-project-name
```

5. Install dependencies:

```
npm install
```

6. Start the dev server:

```
npm run dev
```

Version Control

- We'll use **Git** and **GitHub**
- Track changes to code, collaborate with others
- Key concepts:
 - Repository
 - Commit / Push / Pull
 - Branches & Merge
- GitHub hosts our repositories in the cloud

Database

- We'll use **PostgreSQL** — a powerful open-source SQL database
- Stores and retrieves structured data
- We'll interact with it using SQL queries
- We will use it in Docker for local development

Docker

- Docker is a tool that allows you to create, deploy, and run applications in containers.
- Containers are lightweight, portable, and self-contained environments that can run on any machine.
- Get Docker: docker.com/get-started
- To confirm it's installed, run `docker --version` and `docker run hello-world`

Resources

- [MDN HTTP Guide](#)
- [HTTP Crash Course](#)

Exercise 00

- Setup the development environment in your local machine.
- Create a GitHub repository named `web-system-development` with the following structure:
- Add a README.md file to the repository.

```
/
├── exercises/
│   ├── 01/
│   │   └── index.html
│   └── 02/
│       └── index.html
└── project/
```