

# Web Designing

Static Web Page

Ronak N. Ahir (TA14)

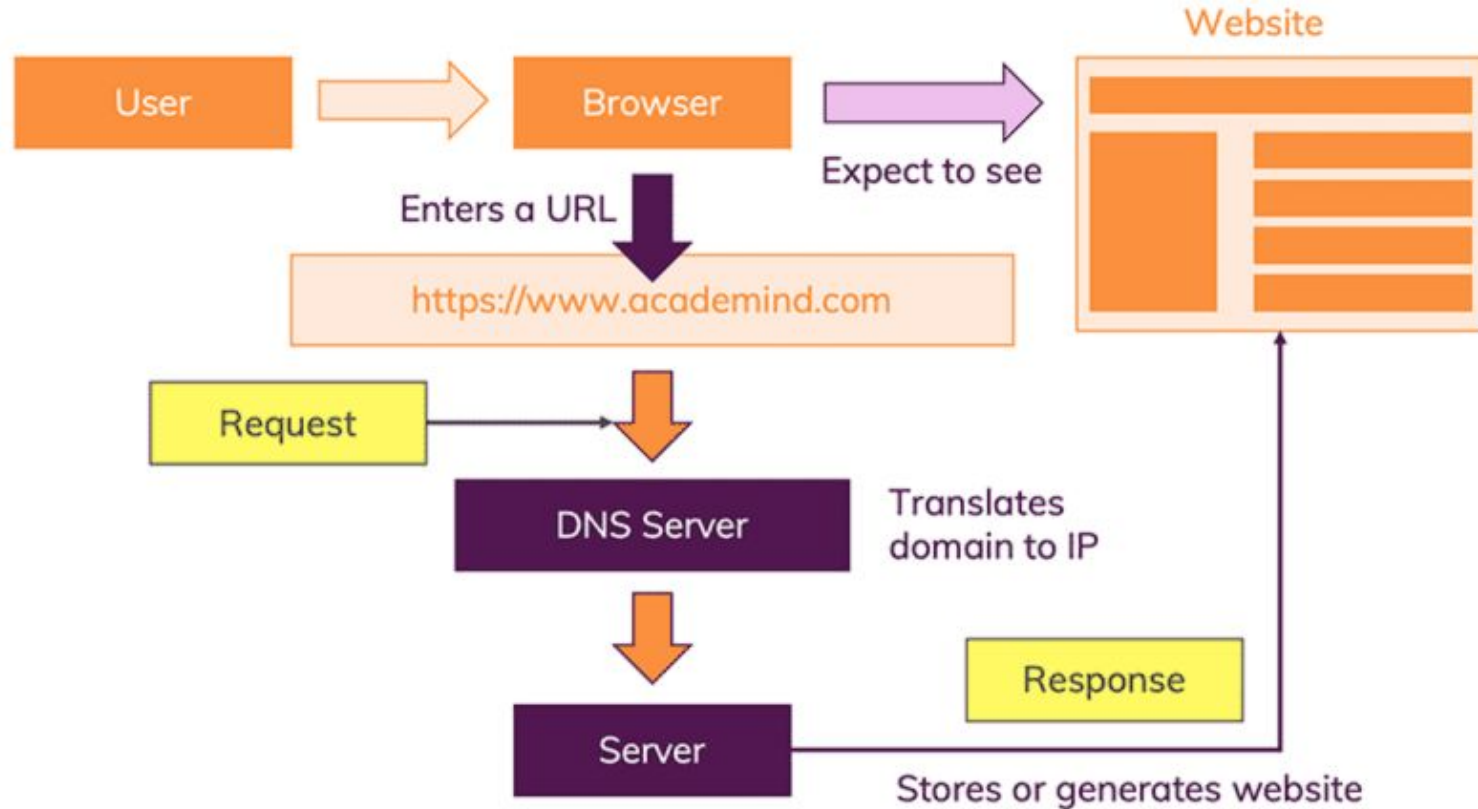
# How Websites Work

Let's start with the most obvious way of using the internet: Visiting a website  
**svnit.ac.in**

List of processes gets executed after hitting Enter.

- The URL gets resolved
- A Request is sent to the server of the website
- The response of the server is parsed
- The page is rendered and displayed

# Overview Execution Flow



# Step 1 - URL Gets Resolved

- The **website code** is obviously not stored on your machine and hence **needs to be fetched from another computer** where it is stored.
- This "other computer" is called a "**server**". Because it serves some purpose, in our case, it serves the website.
- "svnit.ac.in" is called "**a domain**" but actually, the server which hosts the source code of a website, is identified via **IP (Internet Protocol) addresses**.
- The browser sends a "**request**" to the server with the IP address you entered (indirectly - user entered "svnit.ac.in").

# Domain

In reality, we often enter "**svnit.ac.in/cse**" like that. "**svnit.ac.in**" is the domain, "**/cse**" is the path.

Together, they make up the "**URL**" ("**Uniform Resource Locator**").

In addition, you can visit most websites via "www.svnit.ac.in" or just "svnit.ac.in".

Technically, "www" is a subdomain but most websites simply redirect traffic to "www" to the main page.

# IP Address

An IP address typically looks like this: 172.56.180.5

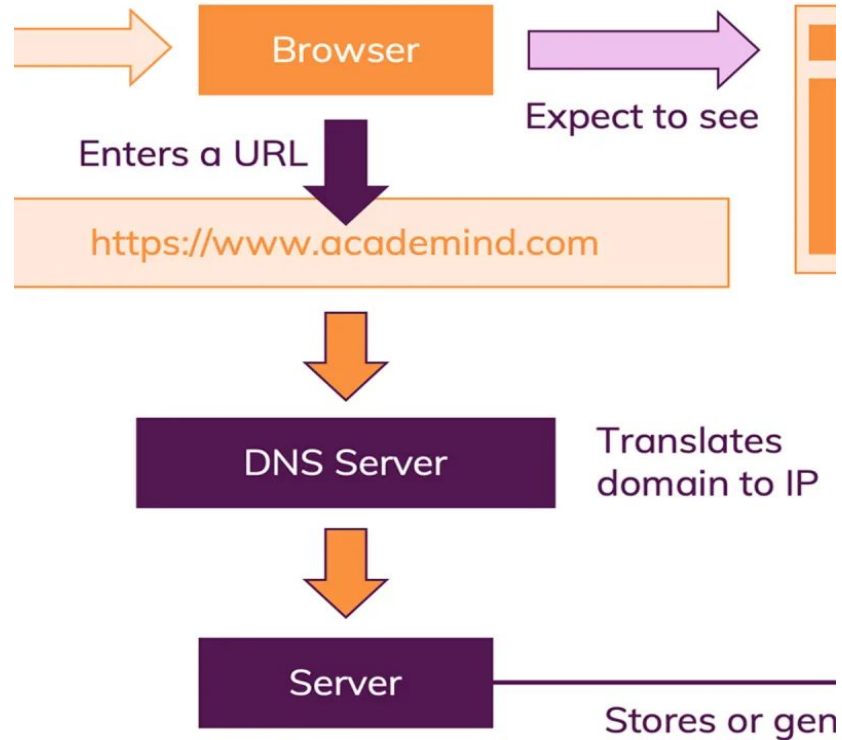
More "modern" form of IP Address is called IPv6.

How is the **domain "svnit.ac.in"** translated to its IP address?

- Special type of server in the internet called "**name server**" or "**DNS server**" (where DNS = "Domain Name System").
- DNS servers **translate domains to IP addresses**. You can imagine those servers as huge dictionaries that store translation tables: Domain => IP address.

When you enter "**svnit.ac.in**", the **browser therefore first fetches the IP address from such a DNS server**.

# DNS Server



## Step 2 - Request Is Sent

With the IP address resolved, the **browser** goes ahead and **makes a request to the server with that IP address**.

"A request" prepared by the browser. The browser bundles up a bunch of information that may contains:

- What's the exact URL?
- Which kind of request should be made?
- Should metadata be attached

and sends that data package to the IP address.



# Request Example:

The data is sent via the "**HyperText Transfer Protocol**" (known as "**HTTP**") - a standardized protocol which defines what a request (and response) has to look like, which data may be included (and in which form) and how the request will be submitted.

## ▼ General

**Request URL:** https://academind.com/

**Request Method:** GET

**Status Code:** 🟢 200

**Remote Address:** 99.84.92.105:443

**Referrer Policy:** no-referrer-when-downgrade

---

## ► Response Headers (11)

## ▼ Request Headers

**:authority:** academind.com

**:method:** GET

**:path:** /

**:scheme:** https

**accept:** text/html,application/xhtml+xml,application/javascript;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.7

**accept-encoding:** gzip, deflate, br

---

# Request

A full URL actually looks like this: `http://svnit.ac.in.com`. The browser **auto-completes** it for you.

And there also is HTTPS - it's like HTTP but encrypted. Most modern pages use that instead of HTTP. A full URL then becomes: <https://svnit.ac.in>

Since the whole process and format is standardized, there is no guessing about how that request has to be read by the server.

# Response

The server handles the request appropriately and returns a "response".

A "response" is a technical thing and kind of similar to a "request". We could say it's basically a "request" in the opposite direction.

Like a request, **a response can contain data, metadata etc.** When requesting a page like `svnit.ac.in`, **the response will contain the code that is required to render the page onto the screen.**

# Example

## ▼ Response Headers

cache-control: public,max-age=0,must-reval  
content-encoding: gzip  
content-type: text/html  
date: Wed, 10 Apr 2019 14:34:56 GMT  
last-modified: Wed, 10 Apr 2019 13:31:31 GM  
server: AmazonS3  
status: 200  
vary: Accept-Encoding  
via: 1.1 eaa1b95207b7e17a6ad05a7c45014762.  
x-amz-cf-id: P82VKqBSkaGF7vCUoornDo4BH\_-\_U  
x-cache: RefreshHit from cloudfront

## Response

## Cookies

## Timing

```
<path class="logo-ora
</svg>
</a>
<span class="spacer"></span>
<button class="hamburger__but
  <span class="hamburger__i
</button>
</div>
<div class="mobile-menu"></div>
</header>
</header>
<main class="layout__content">
  <div class="hero">
    <div class="hero__headline">
      <div class="hero__promo">
        <h1 class="hero__promo-ma
        <h3 class="hero__promo-sul
        <div class="register-wrap
          <input name="newlett
```

# What happens on the server?

That's defined by **web developers**.

In the end, a response has to be sent. That response doesn't have to contain "a website". It can **contain any data - including files or images**.

Some servers are programmed to generate websites dynamically based on the request (e.g. a profile page that contains your personal data), other servers return pre-generated HTML pages (e.g. a news page). Or both is done - for different parts of a webpage.

There also is a third alternative: Websites that are pre-generated but that change their appearance and data in the browser.

## Step 3 - Response Is Parsed

The **browser receives the response sent by the server**. This alone, doesn't display anything on the screen though.

Instead, the next step is that the **browser parses the response**.

The browser checks the data and metadata that's enclosed in the response. And based on that, it decides what to do.

You might've had cases where a PDF opened in your browser. That happened because the response informed the browser that the data is not a website but a PDF document instead. And the browser tries to pick the best handling mechanism for any data type it detects.

In case of website, the response would contain a specific piece of metadata, that tells the browser that the response data is of type text/html.

content-encoding: gzip

content-type: text/html

date: Wed, 10 Apr 2019 14:34:10

This allows the browser to then parse the actual data that's attached to the response as HTML code.

HTML is the core "programming language" (technically, it's not a programming language - you can't write any logic with it) of the web. HTML stands for "Hyper Text Markup Language" and it describes the structure of a webpage.

# Code Example

The code looks like this:

```
<h1>Breaking News!</h1>
```

```
<p>Websites work because browser understand HTML!</p>
```

`<h1>` and `<p>` are called **"HTML tags"**.

**Every HTML tag has some semantic meaning which the browser understands**, because HTML is also standardized. Hence there is no guessing about what a `<h1>` tag means.

The browser knows how to parse HTML and now simply goes through the entire response data (also called "the response body") to render the website.



## Step 4 - Page Is Displayed

The browser goes through the HTML data returned by the server and **builds a website** based on that.

**HTML does not include any instructions** regarding what the site should **look like**

Example: how it should be styled.

**It really only defines the structure** and tells the browser which content is a heading, which content is an image, which content is a paragraph etc.

This is especially important for accessibility - screen readers get all the useful information out of the HTML structure.

A page that only includes HTML

## Welcome to My Homepage

Use the menu to select different Stylesheets

- [Stylesheet 1](#)
- [Stylesheet 2](#)
- [Stylesheet 3](#)
- [Stylesheet 4](#)
- [No Stylesheet](#)

## Same Page Different Stylesheets

This is a demonstration of how different stylesheets can change the layout of your HTML page. different stylesheets in the menu, or by selecting one of the following links:

[Stylesheet1](#), [Stylesheet2](#), [Stylesheet3](#), [Stylesheet4](#).

# CSS

Web page containing only HTML looks not that beautiful !

That's why there's another important technology **CSS** ("Cascading Style Sheets").

**CSS is all about adding styling to the website.** That is done via "**CSS rules**":

Example:

```
h1 {  
    color: blue;  
}
```

This rule would color all `<h1>` tags blue.

# CSS

CSS Rules can be added inside of the HTML code but typically, they're part of **separate .css files** which are requested separately.

A website can be made up of more than the data of the first response we get.


















In practice, websites fetch a lot of additional data (via additional requests and responses) which are kicked off once the first response arrived.

## *How does that work?*

HTML code of the first response simply contains instructions to fetch more data via new requests - and the browser understands these instructions.

# Fetching of Additional Data

```
<link rel="stylesheet"  
href="/page-styles.css" />
```

Name	Status	Type
 academind.com	200	document
 component---node-mo...	200	script
 styles-edc7da8ba2741...	200	script
 app-10c9e1e6fd32855...	200	script
 webpack-runtime-0d8...	200	script
 path---offline-plugin-a...	200	json
 css?family=Muli:300,7...	200	stylesheet
  css?family=Muli:30...	200	fetch
 analytics.js	200	script
  analytics.js	200	fetch
 pages-manifest-d586d...	200	script
 1-8ff6e487b68be897ee...	200	script
 2-db2daebb0d1fedf68...	200	script
 3-325c965f8e273cc0c...	200	script
 component---src-temp...	200	script

# CSS

Together with CSS, the browser is able to display webpages like this

## Welcome to My Homepage

Use the menu to select different Stylesheets

### Stylesheet 1

Stylesheet 2

Stylesheet 3

Stylesheet 4

No Stylesheet

## Same Page Different Stylesheets

This is a demonstration of how different stylesheets can change the layout of your HTML page. You can change the layout of this page by selecting different stylesheets in the menu, or by selecting one of the following links: [Stylesheet1](#), [Stylesheet2](#), [Stylesheet3](#), [Stylesheet4](#).

## No Styles

This page uses DIV elements to group different sections of the HTML page. Click here to see how the page looks like with no stylesheet:

[No Stylesheet](#).

# JavaScript

There is another programming language involved (really is a programming language!): JavaScript.

It's not always visible but all dynamic content you find on a website (e.g. tabs, overlays etc.) is actually only possible because of JavaScript.

It allows web developers to define **code that runs in the browser** (not on the server), hence JavaScript can be used to change the website while the user is viewing it.

# Conclusion

These are the four steps that are always involved when you enter a page address like `svnit.ac.in` and you thereafter see the website content in your browser.

1. URL Gets Resolved
2. Request Is Sent
3. Response Is Parsed
4. Page Is Displayed



# Server-side

Languages that don't work in the browser but that can run on a normal computer (a server is in the end just a normal computer).

Examples would be: **Node.js**, **PHP**, **Python**

Important: With the exception of PHP, you can also use these programming languages for other purposes than web development.

Node.js is indeed primarily used for server-side programming (though it's technically not limited to that), Python is also very popular for data science and machine learning.

# Browser-side

In the browser, there are exactly three languages/ technologies that are used.

These three browser-side languages are all mandatory to know and understand:

**HTML (for the structure)**

**CSS (for the styling)**

**JavaScript (for dynamic content)**

Thank you !