Today's schedule

- Syllabus
- Course Info
- Browsers! The Internet!
- A little bit about HTML and CSS

Syllabus

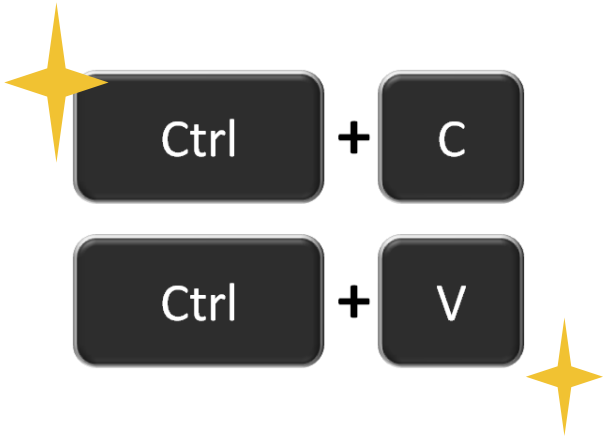
Who are you?

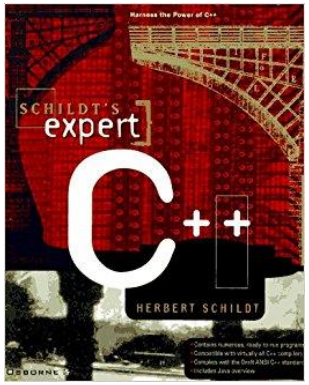
You are:

- A copy/paste programmer of JavaScript, HTML, CSS
  (or you've never used these languages)

- A good programmer in at least one real* programming language
  (Java, C++, etc)

- Frustrated
  (maybe)





In case it's unclear, I'm being facetious

Frustrated?

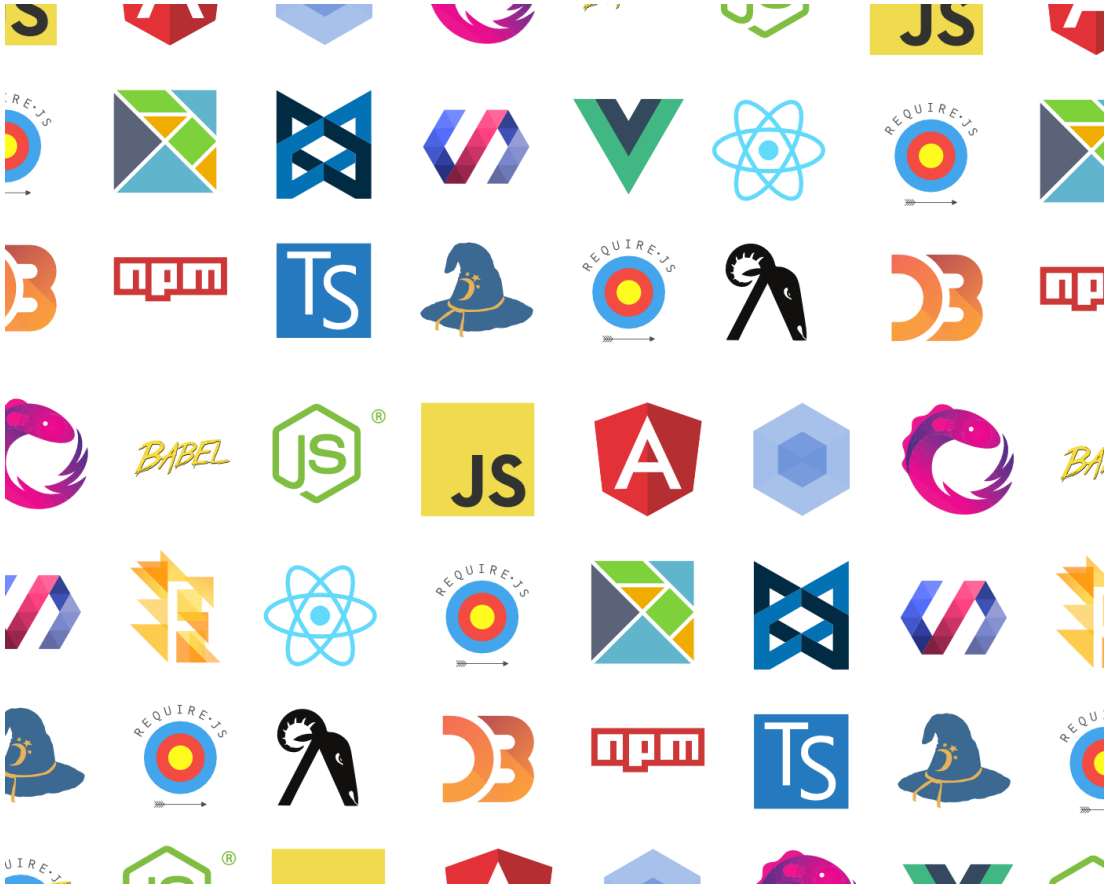
Every beginner CSS tutorial makes CSS look trivially easy:

```
body {
  background-color: red;
}
```

But then when you try to write CSS, literally nothing works:

Frustrated?

You want to learn JavaScript...



...but you're overwhelmed by all the frameworks, libraries, tools, etc and have no idea where to start.

WPR Goals

If you never take another web programming class again, you will leave WPR with the following skills:

- Create **attractive, small scale web sites or apps** that at least mostly work on phones
- Have the **vocabulary and background knowledge** to understand technical writing/discussions about the web (e.g. web API documentation; random blog posts…)

→ Efficent Google **search skill**

- Have the **foundation** to pursue the areas of web programming that you're interested in (if you choose)

(WPR Non-goals)

WPR is **not** a class to take to learn how to code.

WPR is **not** a class that will turn you into a senior frontend/backend developer.

- Nor is any class; software takes years of experience to develop expertise.

WPR is **not** a class that will teach you all there is to know about web programming.

- For example, we will **not** teach how to support old browsers, legacy devices, etc.

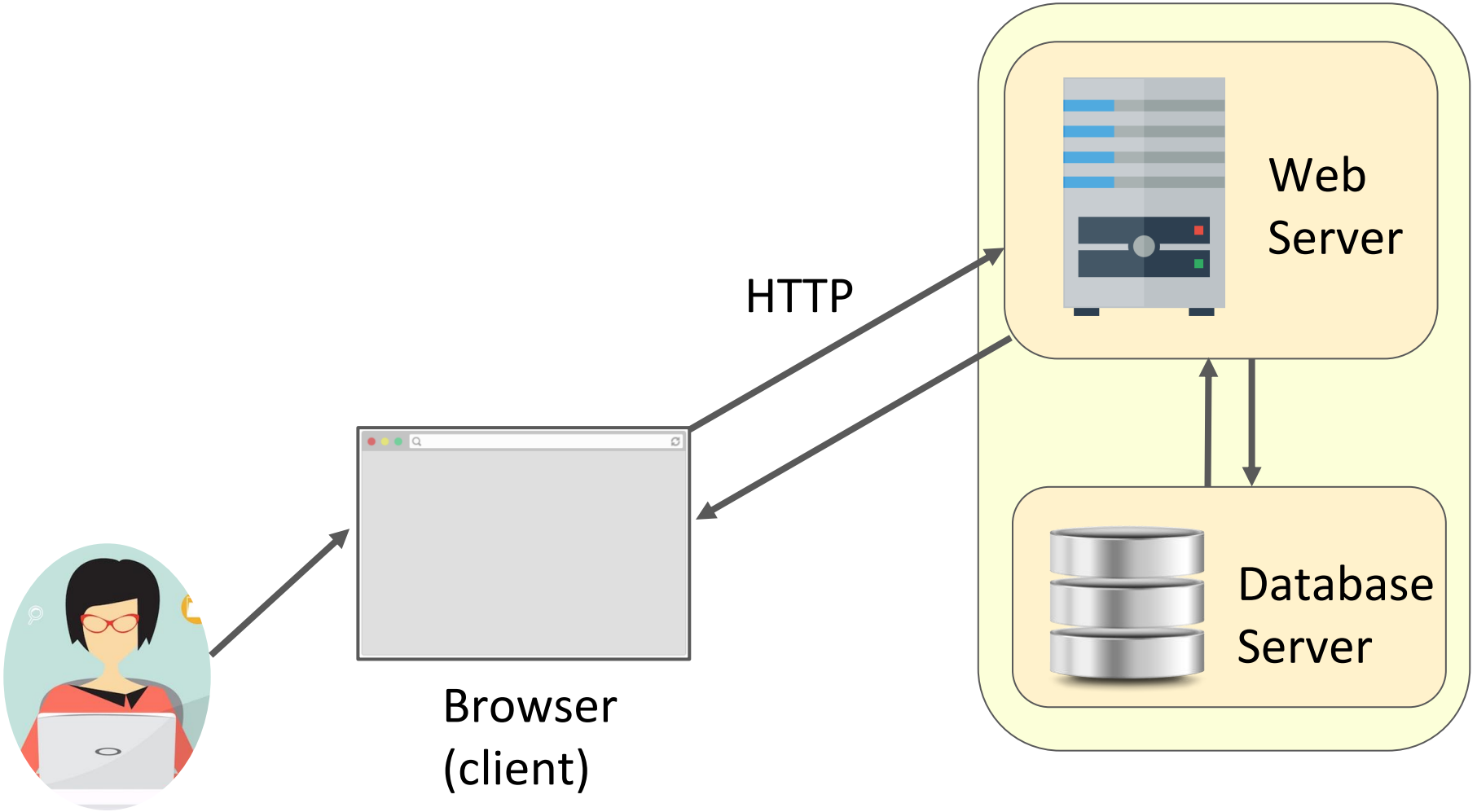
WPR, in detail

- Static website (Review & extend)
  - HTML
  - CSS
  - **JavaScript**
- Backend basics
  - Server on NodeJS and Express
  - Database via MongoDb and Mongoose
- Frontend basics
  - ReactJS

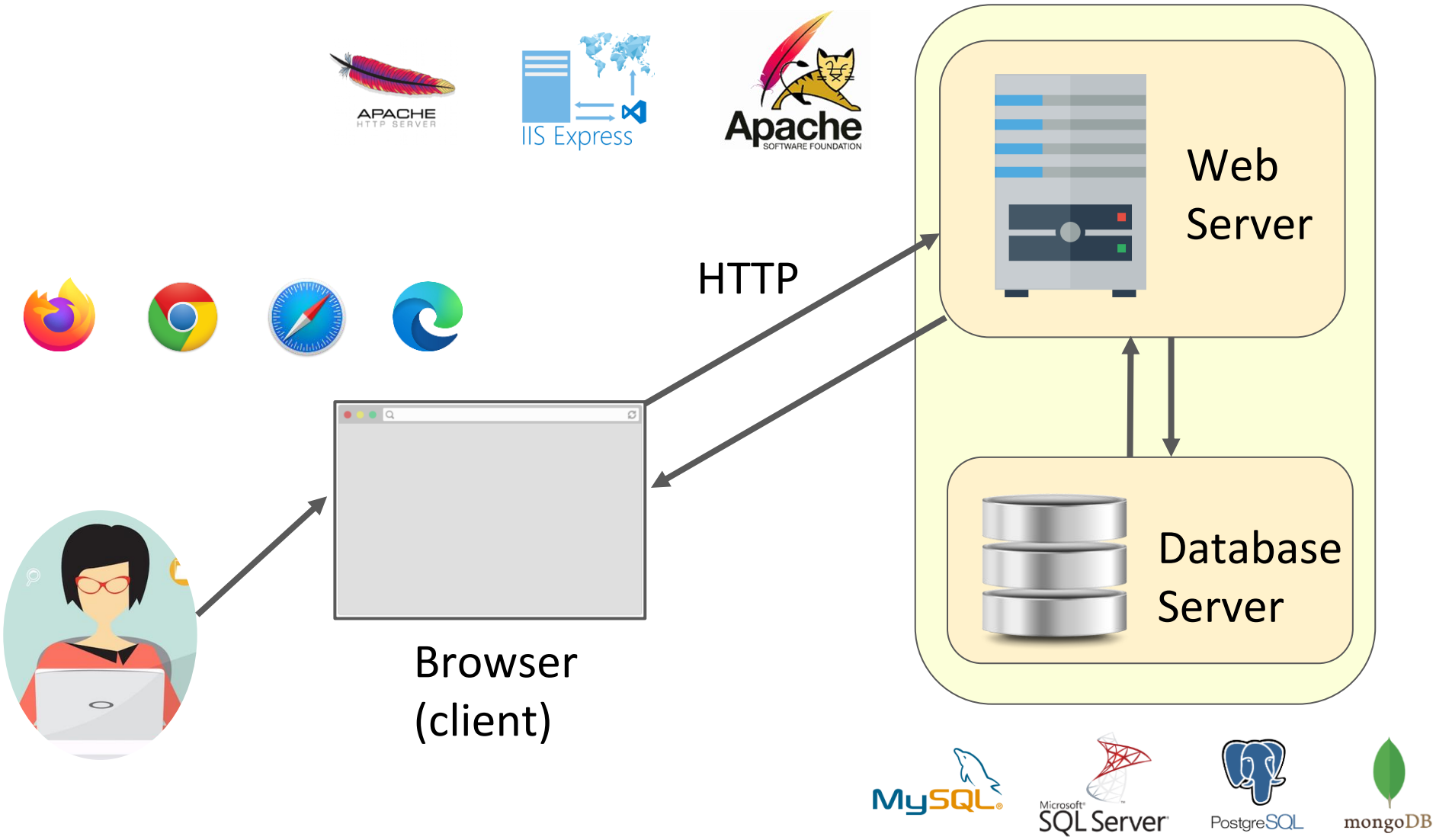
(Uh…

a) How is this an "opinionated" list of topics?)

Web Components

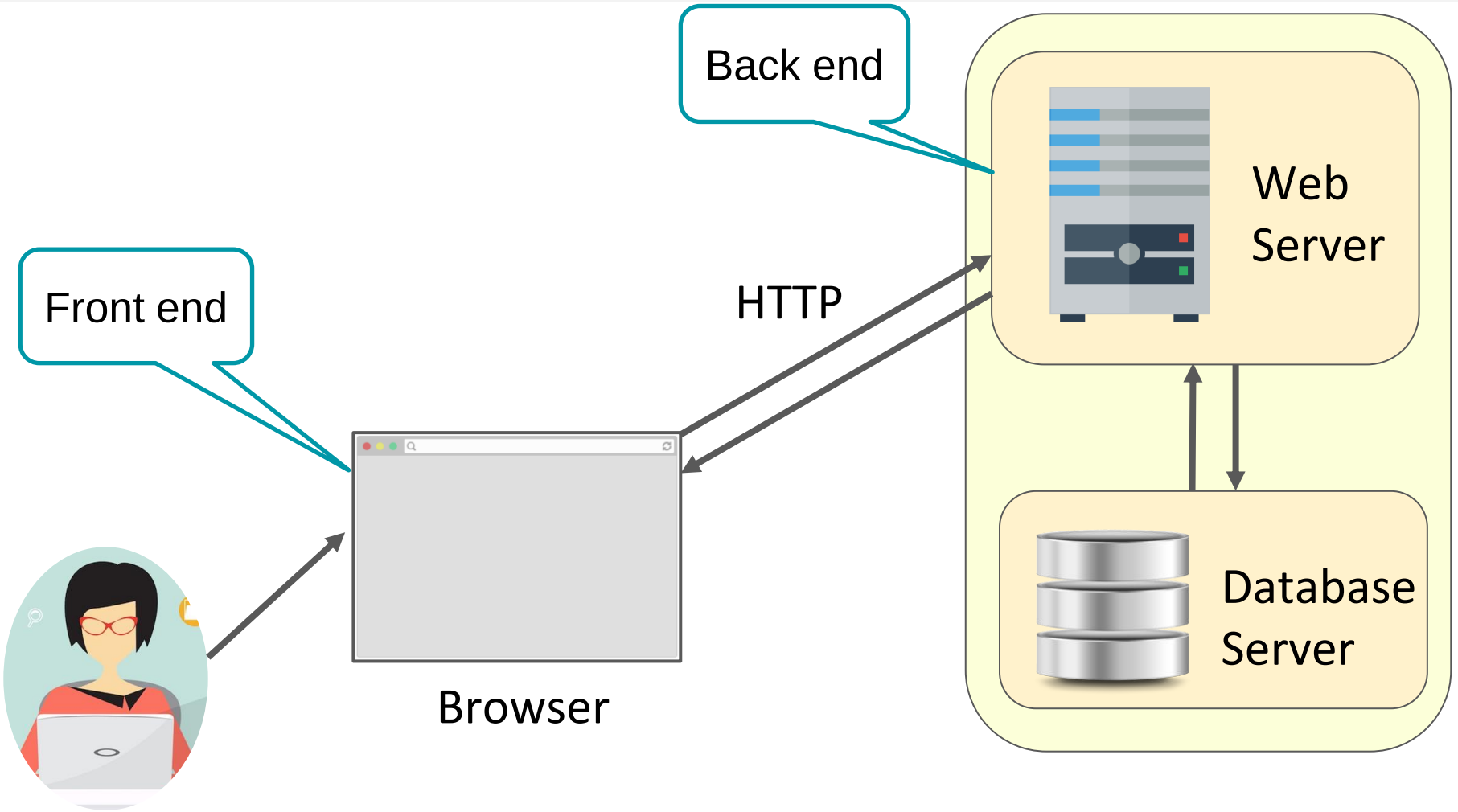


Web Components

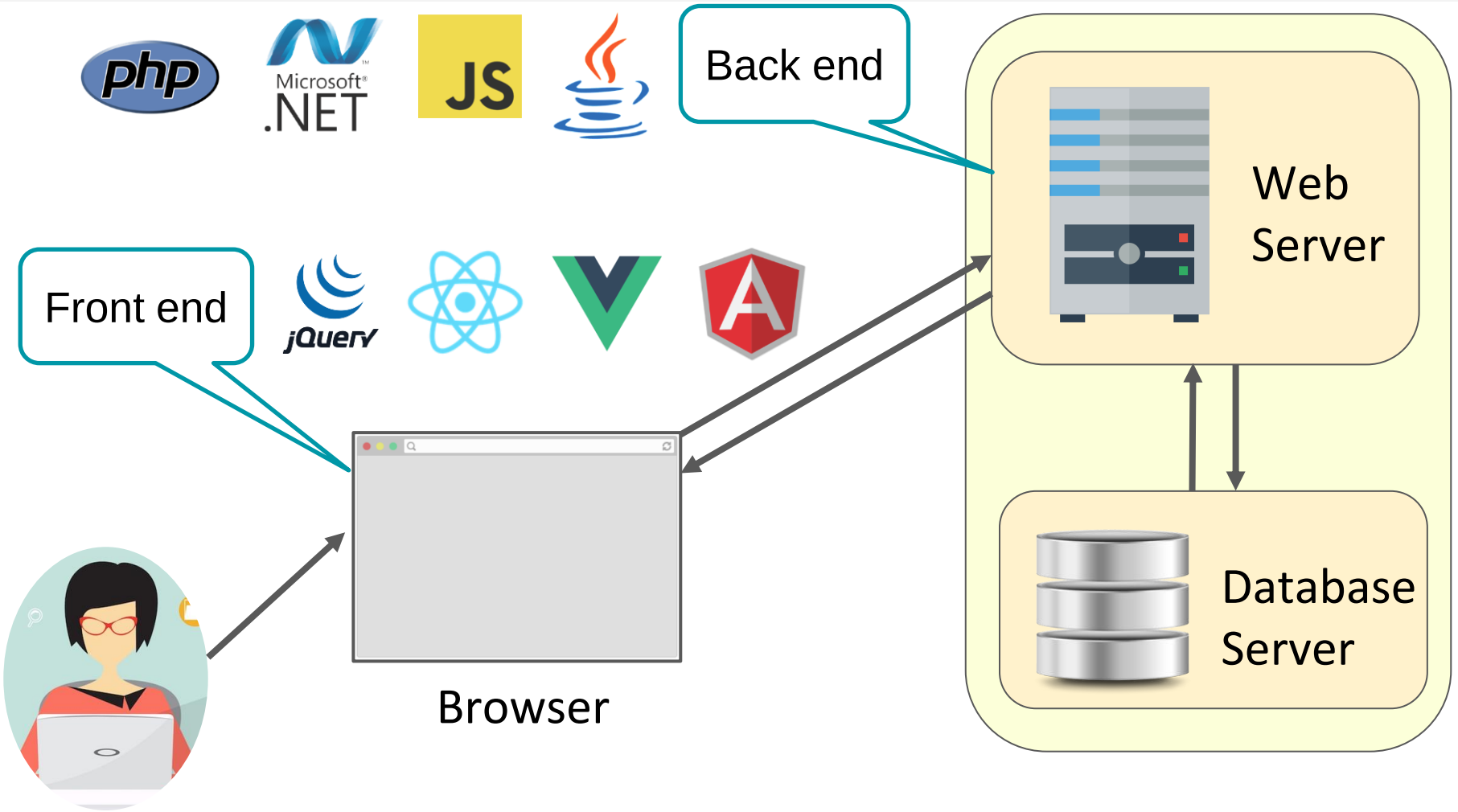


Frontend vs Backend?

Web Components



Web Components



Career positions?

Target career positions?

In web app projects:

Requirements

Web application

- Requirement Analyst / BA
- Designer
- Frontend Developer
- Backend Developer
- Tester
- PM

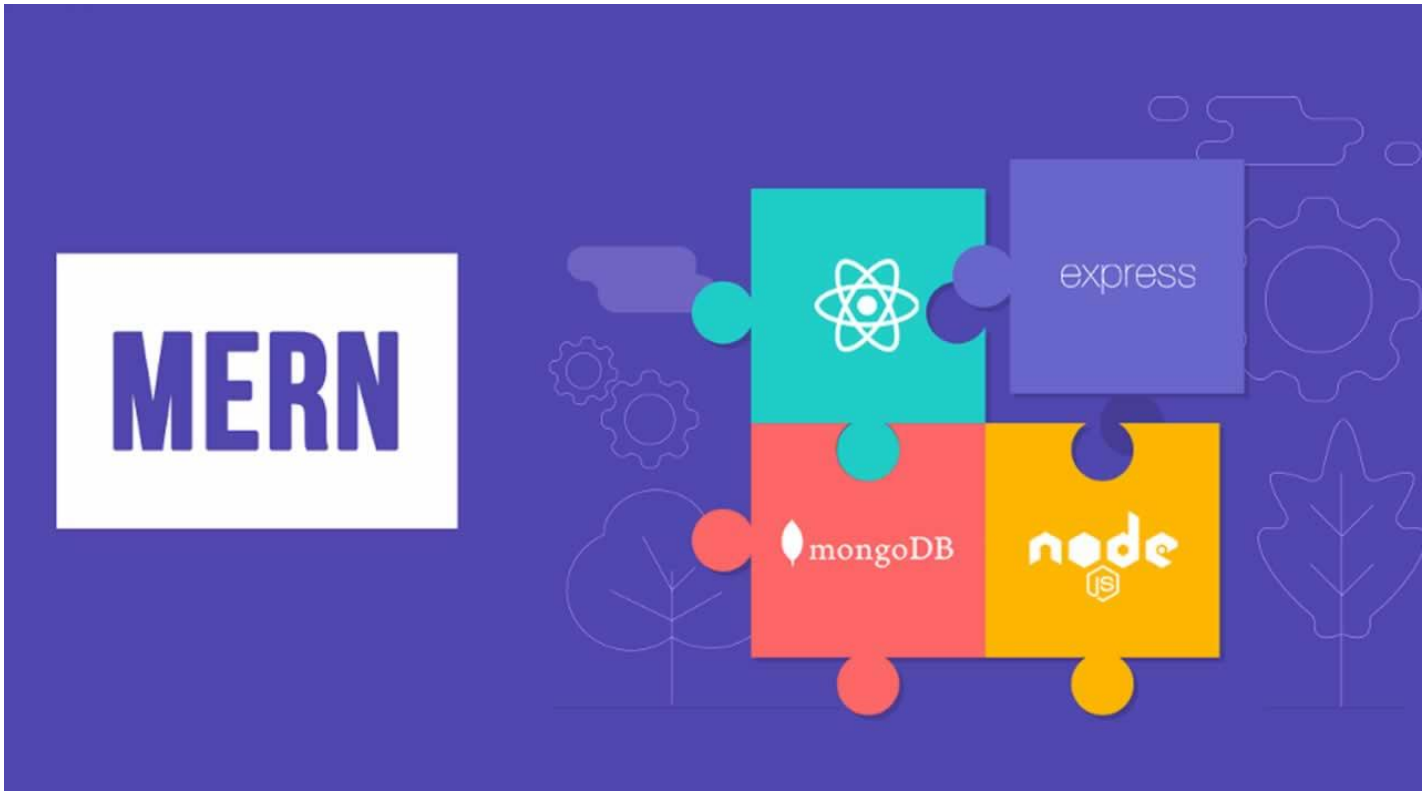
Full Stack Web Developer

WPR Goals

Full Stack JavaScript Developer

MERN stack

(MongoDB – ExpressJS – ReactJS – NodeJS)



WPR: CSS

HTML (~1 week)

- Key concepts: inline, block, inline-block

CSS (~1.5 weeks)

- Multiple rendering styles: natural, flex, positioned, float
- Mobile layouts
- Transforms and animations
- **FYI: No libraries or compiled CSS**

WPR: Modern JS / ES6+

Later in the quarter, we will read and write JavaScript that looks sort of like this:

```
(async () => {
  let choice = 'e';
  do {
    choice = await askQuestion('Enter choice');
    await processChoice(choice);
  } while (choice != 'e');
})();
```

WPR: Modern JS / ES6+

JavaScript (~3 weeks)

- Review JavaScript basic
- JavaScript classes
- Relevant functional programming
  - Lambdas
  - Generator functions and async/await
  - "Fat arrow" vs function
  - Closures
- Creating and using Promises
- Understanding the Event Loop
- Modules and encapsulation

WPR : Baby's first backend

WPR coverage of server-side programming will be light.

Backend stack:

NodeJS + Express + MongoDB via Mongoose (~4 weeks)

- What is a server
- What is npm
- How to serve static web pages
- How to serve JSON via REST APIs
- Writing to and loading from a database

WPR : Baby's first frontend

WPR coverage of client-side programming will be light.

Frontend stack:

ReactJS (~3 weeks)

- Organizing program using components
- Creating React components
- How to handle events (e.g. user interactions)
- Fetching data from APIs

Related courses

DBS :

Content: Database design

Related: Creating website using PHP + MySQL

SE1:

Content: Software development life cycle

Related: Applying software development life cycle in a web app project

Course info

Disclaimer

This is the ~~first ever~~ ~~second~~ third offering of WPR, meaning:

- **Everything is subject to change.** Including everything I've just told you and everything I'm about to tell you.

- **There will be all the mistakes of a new course!**
  - Bugs in homework
  - Awkward lectures
  - Things that are too hard / too easy

Please be patient with us! We are also soliciting your constructive feedback.

Grades

Attendance: 10%

Assignments: 30%

Final Exam: 60%

- **Assignments**: 3 assignments with 10%/each.
- **Final Exam**: Multiple choice.

Lateness policy

- Every homework/quiz may be submitted up to 48 hours after the deadline, without penalty.

- Homework/quiz submitted on time will receive a small bonus to assignment score & **for consideration**.

- The assignments must be submitted on time. Max 3 days late with 10% mark deduction per day.

Browser and Text editor/IDE

- **Text editor:** You can use whatever you want. We recommend Visual Studio Code.

- **Browser:** Your code must work on Firefox Developer, as that is what your TAs will use when grading your assignment. It will not be tested in any other browser.

Lectures

- Nothing will be graded in lecture
- But please come!
- If you attend and do not feel the lectures are helpful, please kindly tell us why :) we will have a feedback link up soon!

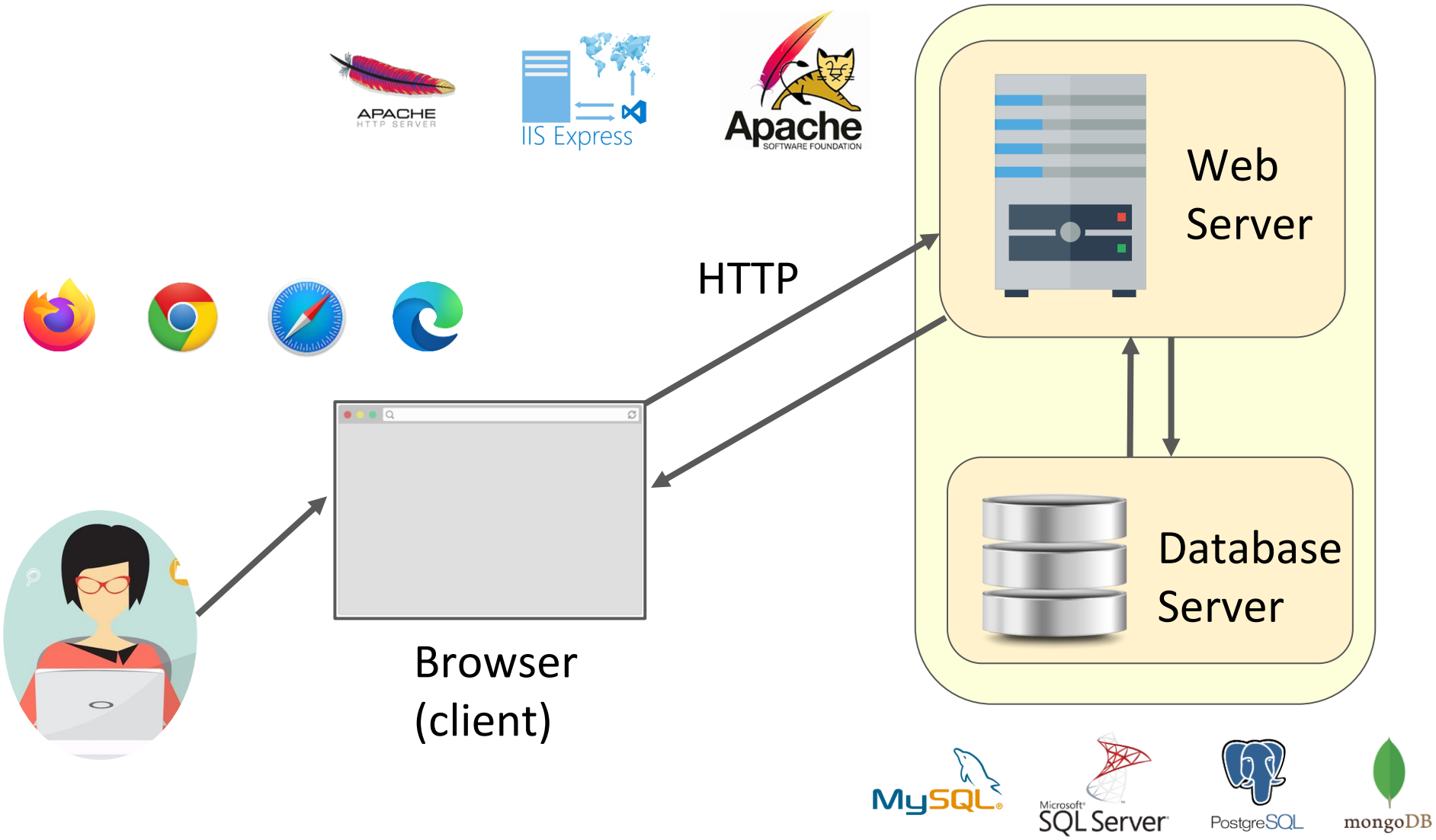
Questions?

Today's schedule

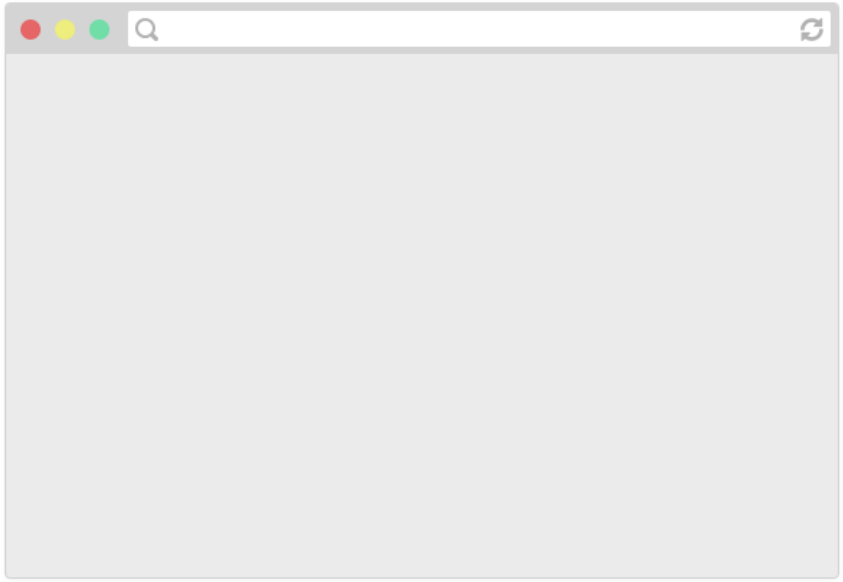
- ~~Syllabus~~
- ~~Course Info~~
- Browsers! The Internet!
- A little bit about HTML and CSS

Browsers!
The Internet!
The web!

Web Components

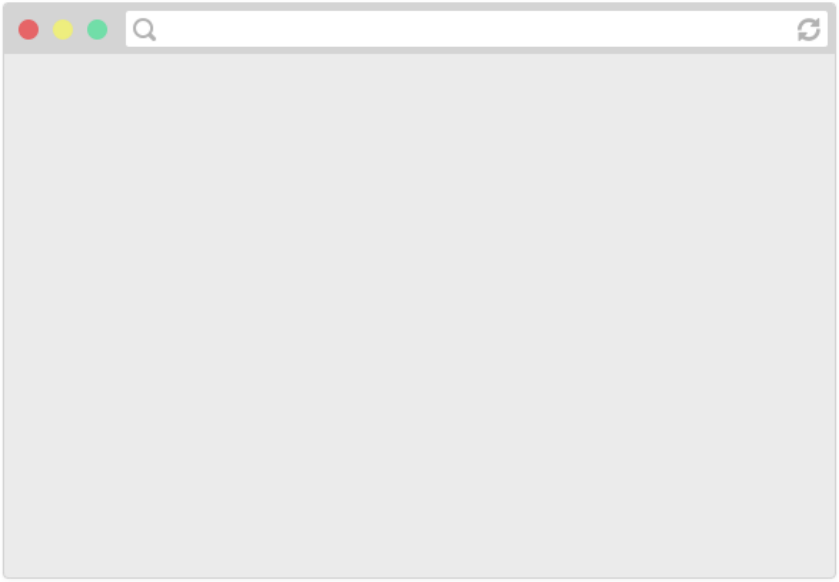


How do web pages work?



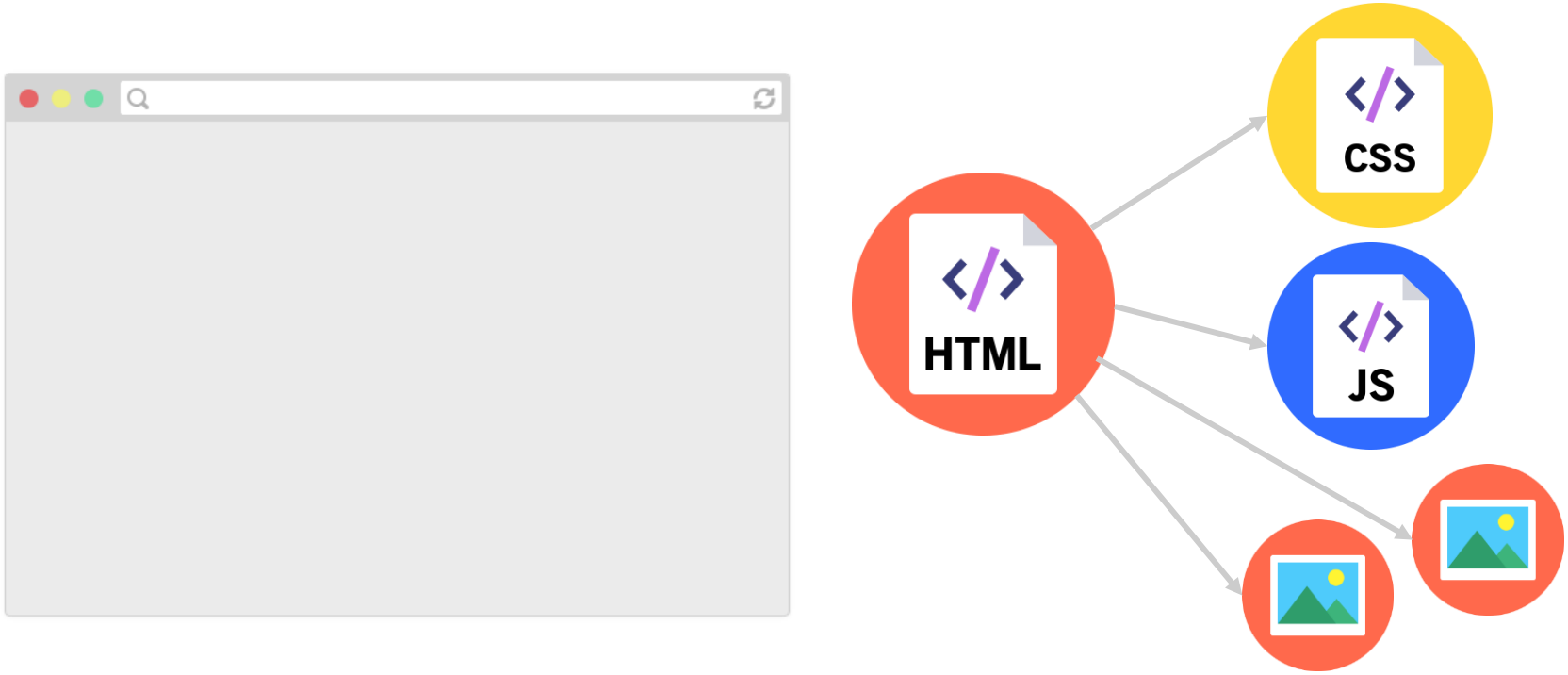
Browsers are applications that can display web pages.
E.g. Chrome, Firefox, Safari, Internet Explorer, Edge, etc.

How do web pages work?



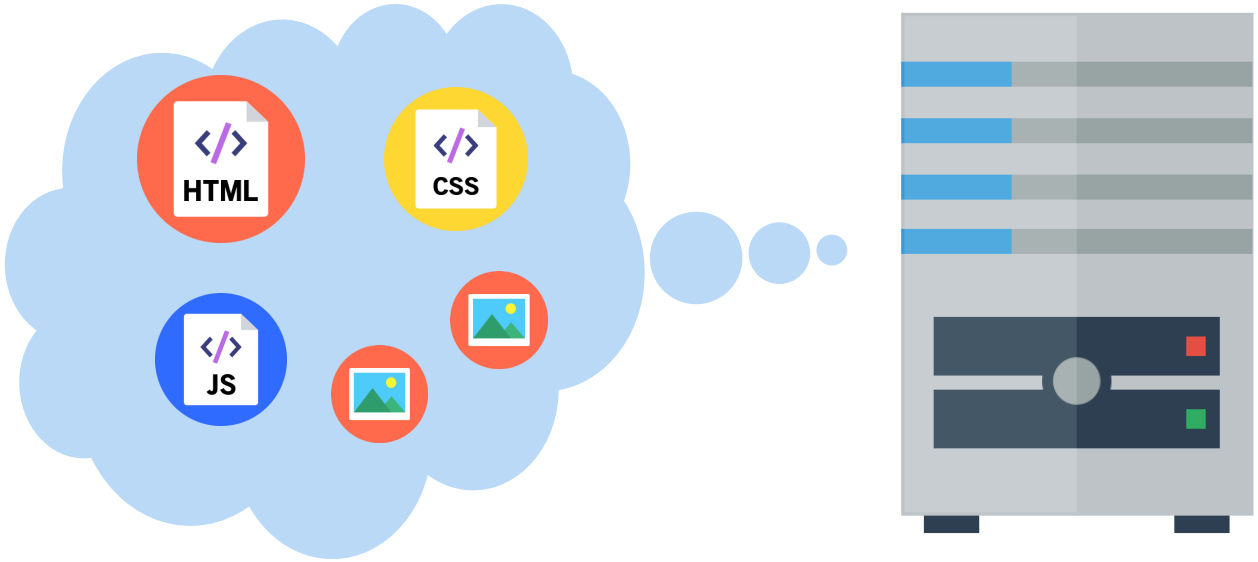
Web pages are written in a markup language called **HTML**, so browsers display a web page by reading and interpreting its HTML.

How do web pages work?



The HTML file might link to other resources, like images, videos, as well as **JavaScript** and **CSS** (stylesheet) files, which the browser then also loads.

How do web pages work?

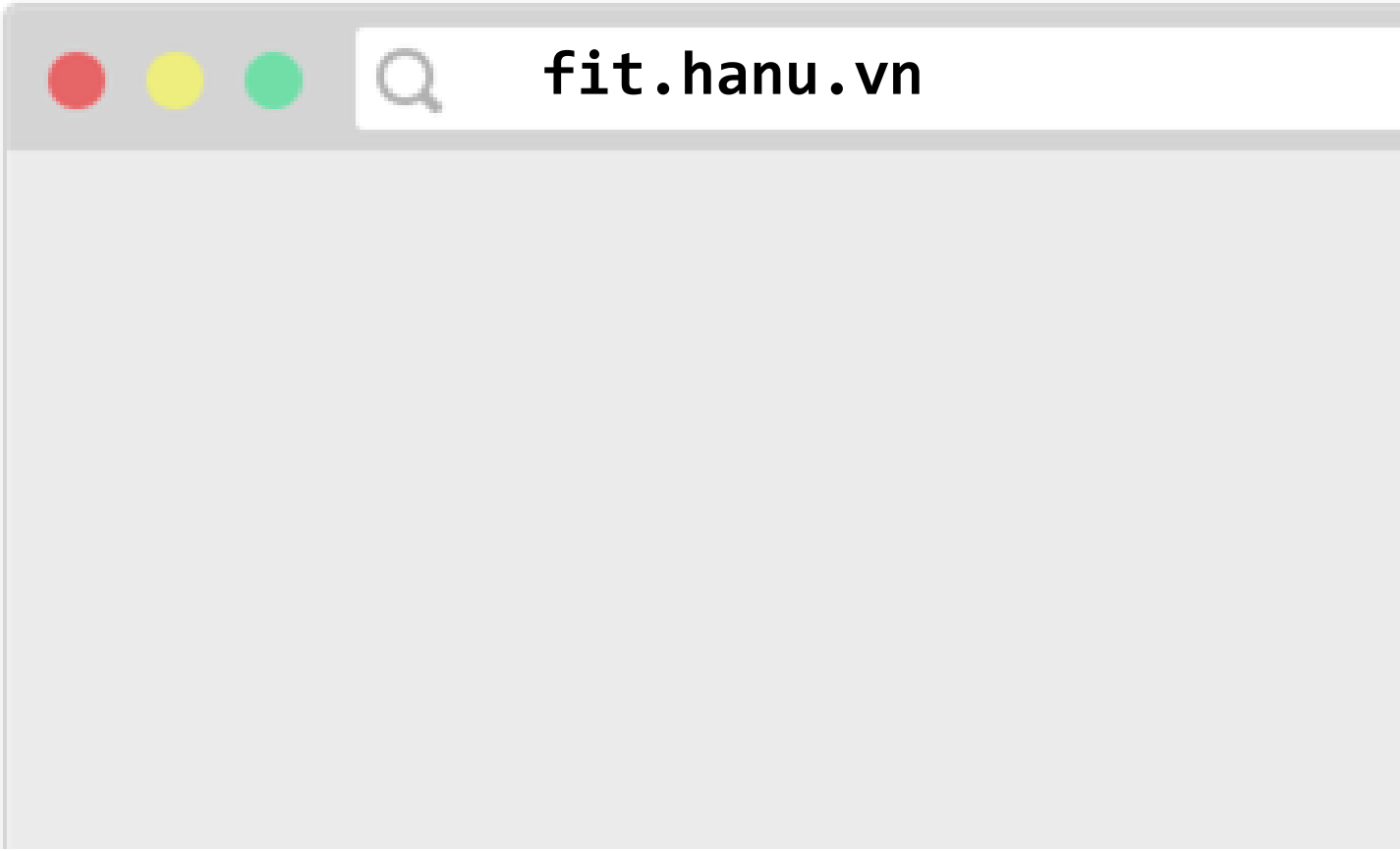


A **web server** is a program running on a computer that delivers web pages in response to requests.

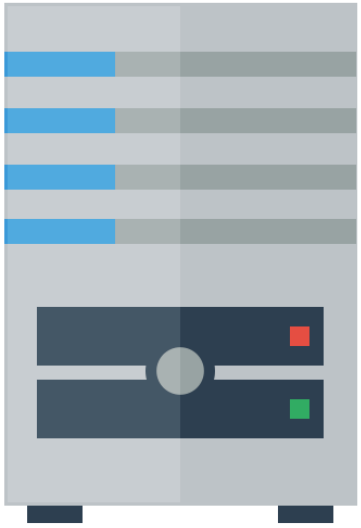
It either stores or generates the web page returned.

How do web pages work?

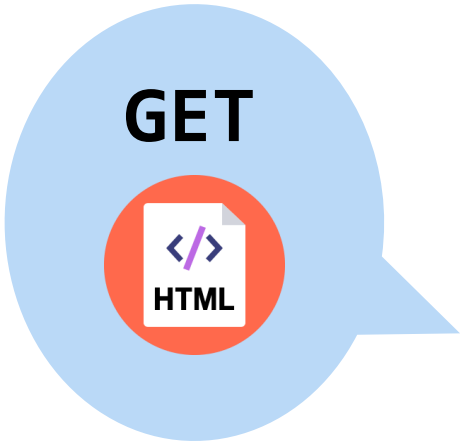
1. You type in a URL, which is the address of the HTML file on the internet.

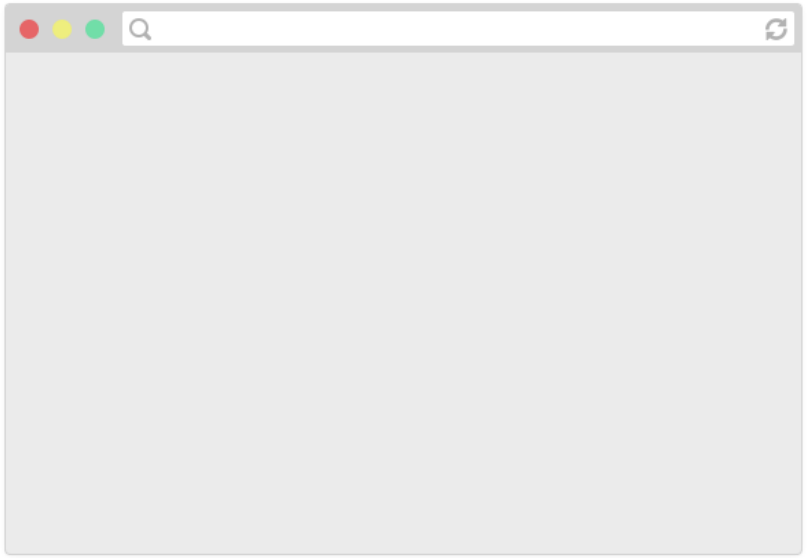


How do web pages work?

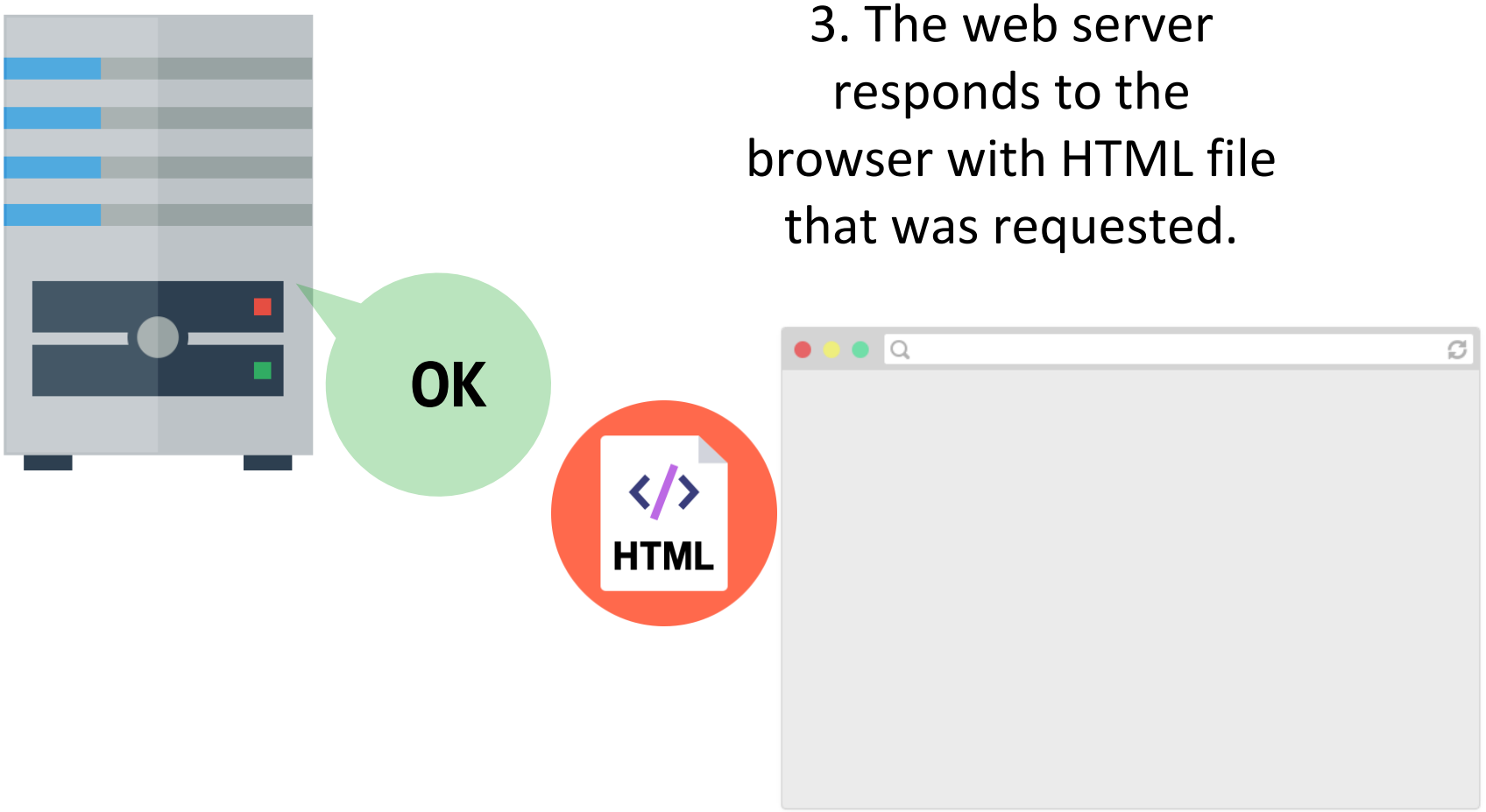


2. The browser asks the web server that hosts the document to send that document.





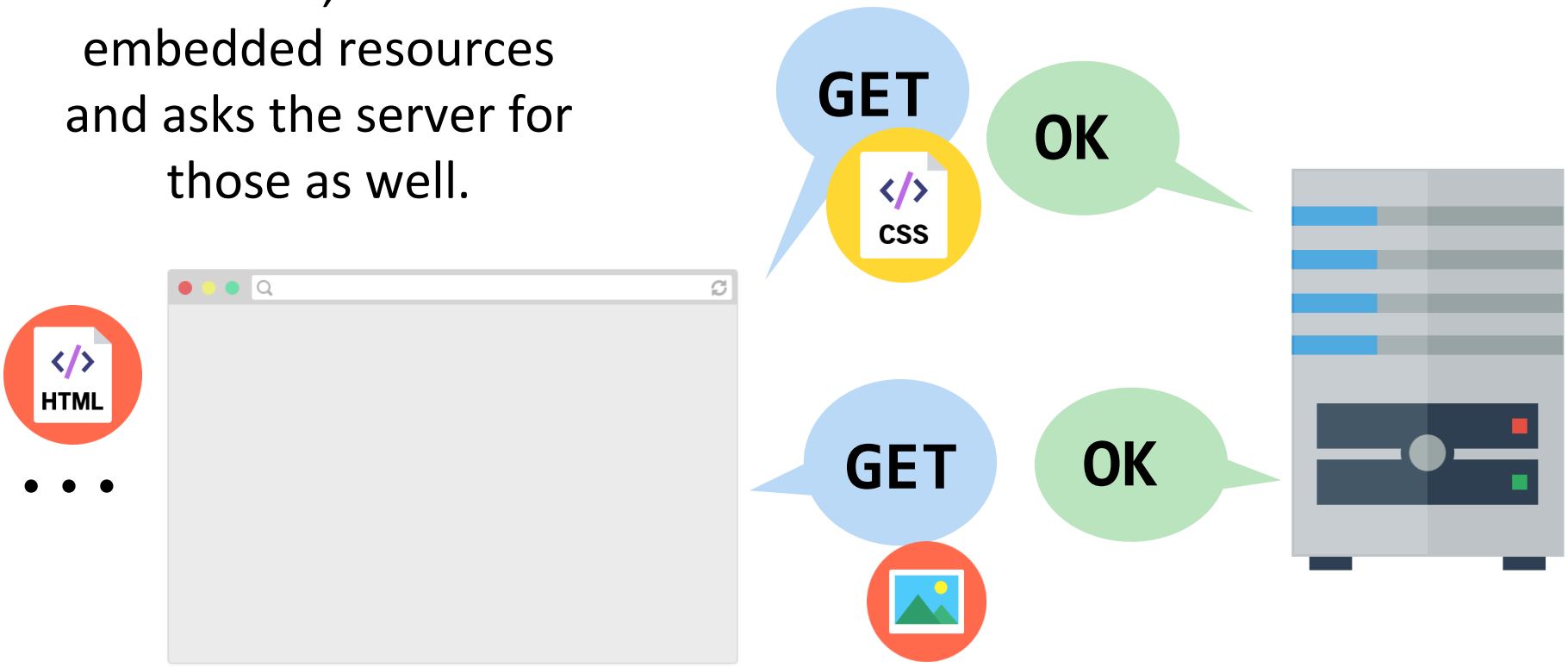
How do web pages work?



3. The web server responds to the browser with HTML file that was requested.

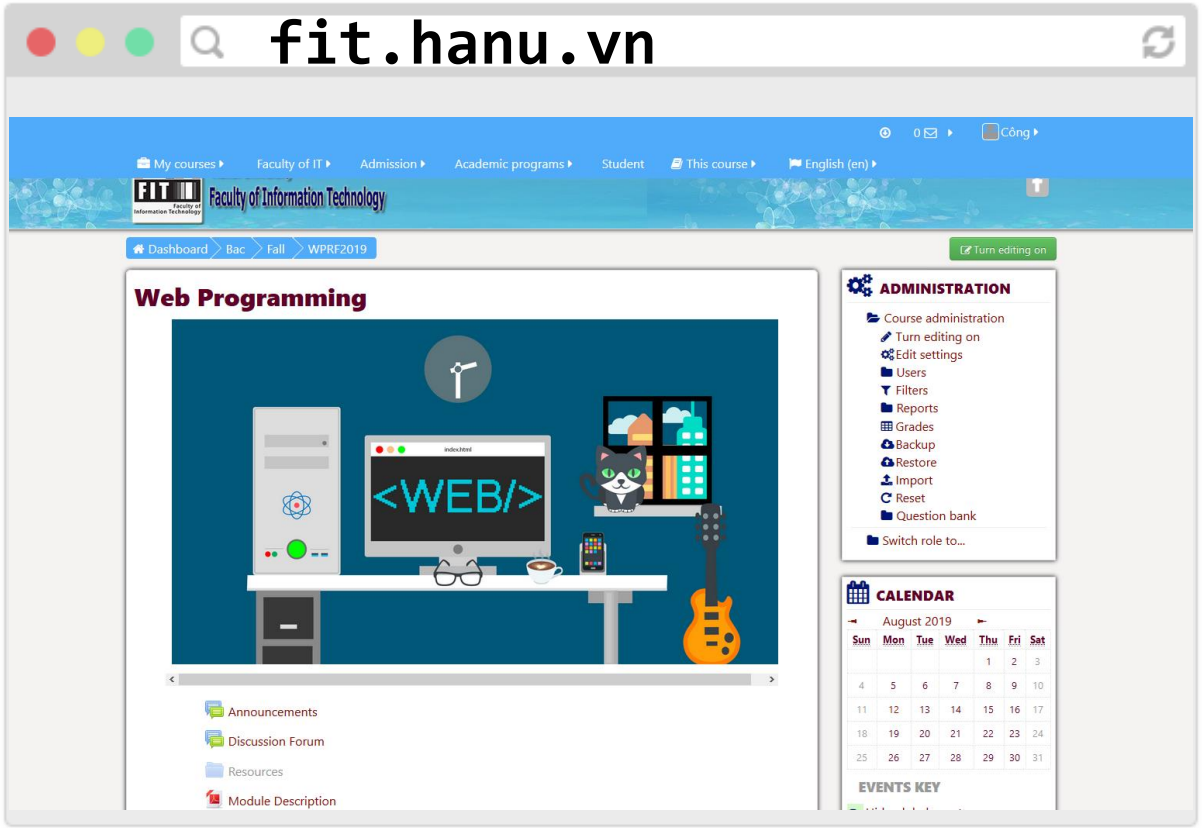
How do web pages work?

4. The browser reads the HTML, sees the embedded resources and asks the server for those as well.



How do web pages work?

5. The web page is loaded when all the resources are fetched and displayed.



P.S.

(That was obviously very hand-wavy. We'll get more detailed when we talk about servers later in the quarter.)

HTML and CSS

HTML and CSS strategy

Assumption: Most people have cursory familiarity with HTML and CSS. Therefore we will:

- **Speed through** the obvious stuff
- **Skip** self-explanatory syntax
- **Skip** the parts you can look up easily through Google

 **Therefore, be aggressive with questions!** 

What is HTML?

HTML (**H**yper**t**ext **M**arkup **L**anguage)

- Describes the **content** and **structure** of a web page; not a programming language.
- Made up of building blocks called **elements**.

```
<p>
  HTML is <em>awesome!!!</em>
  <img src="puppy.png" />
</p>
```

Basic HTML page structure

(i.e. copy/paste boilerplate)

```
<!DOCTYPE html>
<html>
  <head>
   <title>WPR</title>
  </head>

  <body>
   ... contents of the page...
  </body>
</html>
```

Saved in a *filename*`.html` file.

Basic HTML page structure

(i.e. copy/paste boilerplate)

Metadata that doesn't appear in the viewport of the browser

Contents that render in the viewport of the browser

```
<!DOCTYPE html>
<html>
  <head>
    <title>WPR</title>
  </head>

  <body>
    ... contents of the page...
  </body>
</html>
```

E.g. **<title>** shows up as the name of the tab

HTML elements

```
<p>
  HTML is <em>awesome!!!</em>
  <img src="puppy.png" />
</p>
```

- An element usually has start and ending tags (** ** and ** **)
  - **content**: stuff in between start and end tags
- An element can be self-closing (**img**)
- An element can have attributes (**src="puppy.jpg"**)
- Elements can contain other elements (**p** contains **em** and **img**)

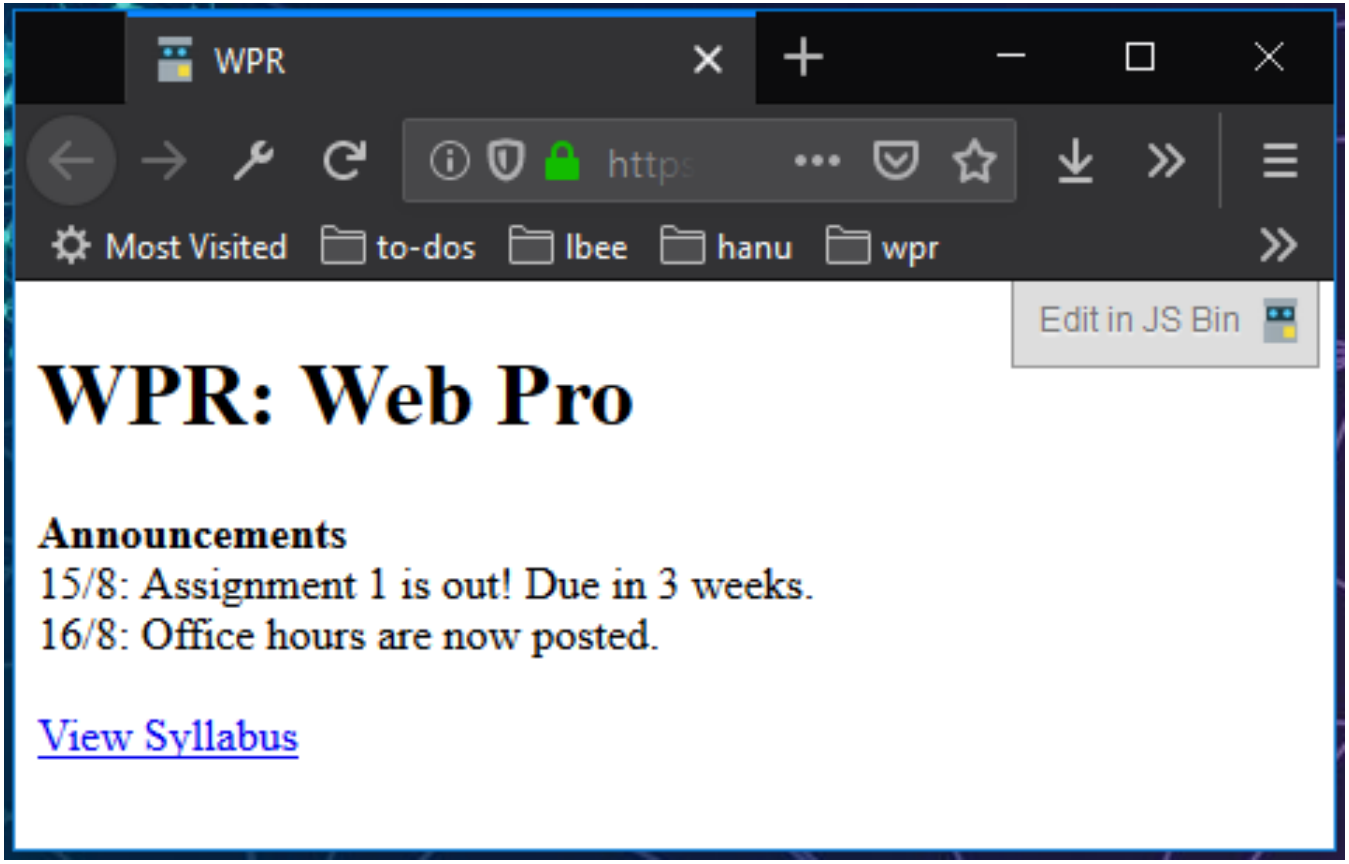
Some HTML elements

(to place within **<body>**)

Top-level heading **h1, h2, … h6**	** **Moby Dick** **
Paragraph	** **Call me Ishmael.** **
Line break	since feeling is first** ** who pays any attention
Image	****
Link	****click here!****
Strong (bold)	****Be BOLD****
Emphasis (italic)	He's my ****brother**** and all

Exercise: Course web page

Let's write some HTML to make the following page:



Exercise: Course web page

HTML boilerplate

```
<!DOCTYPE html>
<html>
  <head>
    <title>WPR Pro</title>
  </head>

  <body>
   ...
  </body>
</html>
```

Plaintext contents of the page

```
WPR: Web Pro

Announcements
15/8: Assignment 1 is out!
Due in 3 weeks.
16/8: Office hours are now
posted.

View Syllabus
```

JSBin

Solution

```
<!DOCTYPE html>
<html>
  <head>
    <title>WPR</title>
  </head>
  <body>
    <h1>WPR: Web Pro</h1>
    <strong>Announcements</strong><br/>
    15/8: Assignment 1 is out! Due in 3 weeks.<br/>
    16/8: Office hours are now posted.<br/>
    <br/>
    <a href=“http://fit.hanu.vn/mod/resource/view.php?id=5778">

      View Syllabus
    </a>
  </body>
</html>
```

That was weird

- We saw that HTML whitespace collapses into one space…

```
<h1>WPR: Web Fun</h1>
<strong>Announcements</strong><br/>
15/08: Assignment 1 is out!<br/>
```

- Except weirdly the **<h1>** heading was on a line of its own, and **** was not.

Hmmm… strange…
Oh well, it works! Let's move on!!!

CSS

CSS

CSS: **C**ascading **S**tyle Sheets

- Describes the **appearance** and **layout** of a web page
- Composed of CSS **rules**, which define sets of styles

```
selector {
  property: value;
}
```

CSS

A CSS file is composed of **style rules**:

```
selector {
  property: value;
}
```

selector: Specifies the HTML element(s) to style.
property: The name of the CSS style.
value: The value for the CSS style.

Saved in a *filename*`.css` file.

CSS

```
// NOT REAL CSS
fork {
   color: gold;
}
```

"All forks on the table should be gold"

CSS

```
p {
    color: blue;
    font-weight: bold;
}
```

"All **<p>** elements on the page should be blue and bold"

Linking CSS in HTML

(i.e. copy/paste boilerplate)

```
<!DOCTYPE html>
<html>
  <head>
    <title>WPR</title>
    <link rel="stylesheet" href="filename.css" />
  </head>

  <body>
   ... contents of the page...
  </body>
</html>
```

Some CSS properties

There are over 500 CSS properties! Here are a few:

Font face (mdn)	`font-family: Helvetica;`
Font color (mdn)	`color: gray;`
Background color (mdn)	`background-color: red;`
Border (mdn)	`border: 3px solid green;`
Text alignment (mdn)	`text-align: center;`

Aside: Mozilla Developer Network (MDN) is the best reference for HTML elements and CSS properties

- The actual W3 spec is very hard to read (meant for browser developers, not web developers)

Main ways to define CSS colors:

140 predefined names (list)

```
color: black;
```

rgb() and rgba()

```
color: rgb(34, 12, 64);
color: rgba(0, 0, 0, 0.5);
```

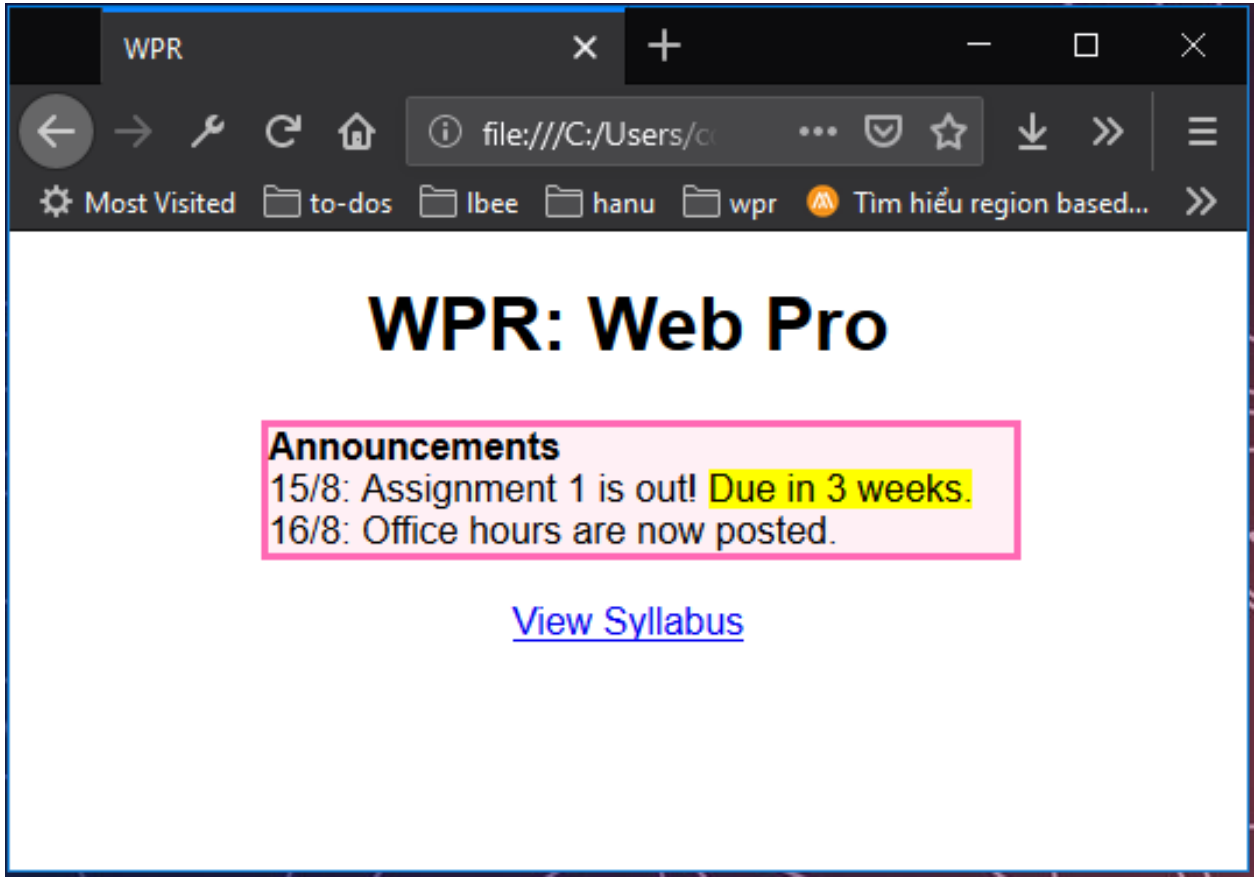
Hex values

```
color: #00ff00;
color: #0f0;
color: #00ff0080;
```

- The "a" stands for **alpha channel** and is a **transparency** value
- Generally prefer more descriptive over less:
  1. Predefined name
  2. rgb / rgba
  3. Hex

Exercise: Course web page

Let's write some CSS to style our page:



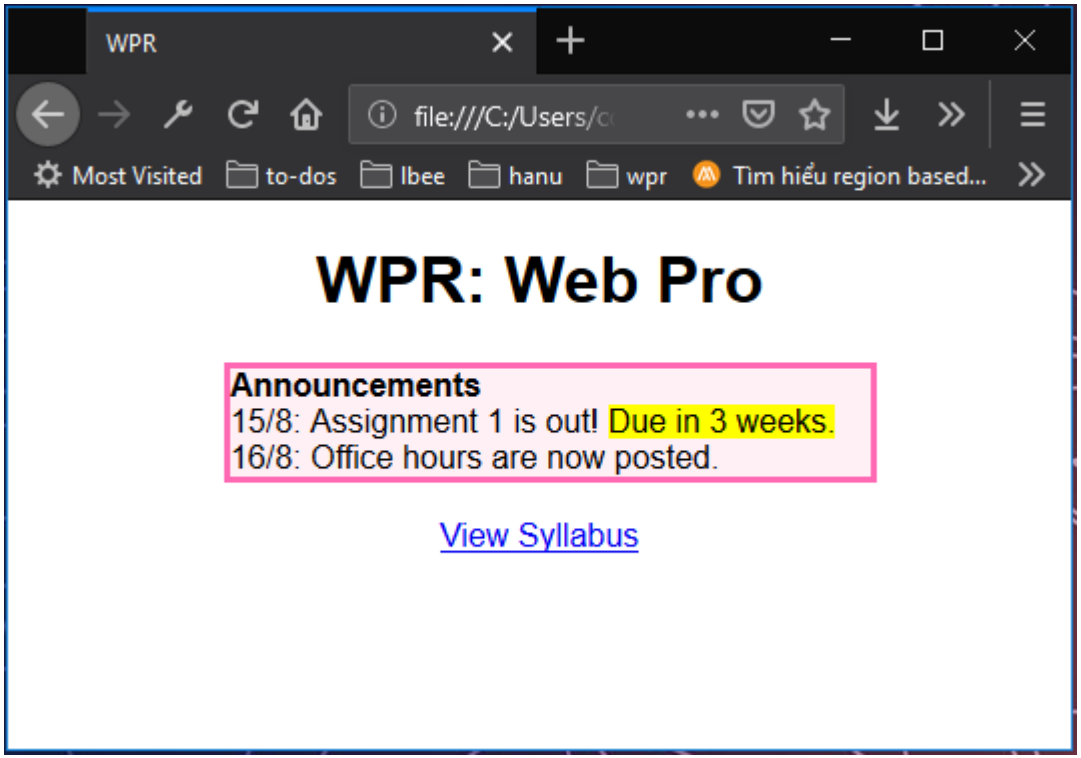
Exercise: Course web page

Let's write some CSS to style our page:

Font face: Helvetica

Border: `hotpink 3px`
Background color: `lavenderblush`
Highlight: `yellow`

- Box is **centered**
- Header and link are **centered**
- Box contents are **left-aligned**



JSBin

CSS exercise debrief

Some **key techniques:**

- Add invisible containers in HTML to select groups of elements in CSS.
- Apply styles to parent / ancestor element to style parent and all its children. (Will talk more about this later.)

But we encountered **more weirdness**...

- Couldn't set `text-align: center;` to the `<a>` or `<strong>` tags directly, but could center `<p>` and `<h1>`
- Had to set a `width` on the box to make it hug the text … any other way to do this?
- How to center the box?! How do you highlight?!

Q: Why is HTML/CSS so weird??

A: There is one crucial set of rules we haven't learned yet...

block vs **inline** display

Next time!