university of groningen

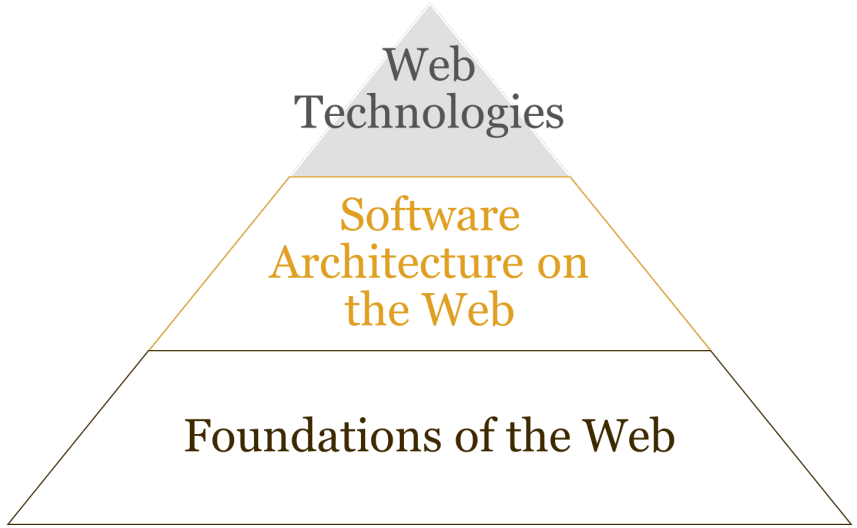
Web Engineering (WBCS008-05)

Set 4: Web Technologies

Vasilios Andrikopoulos
v.andrikopoulos@rug.nl

Outline

- HTML
- CSS
- Static vs Dynamic Pages
  - Server- vs Client-side processing
  - AJAX & FetchAPI
  - SPA vs MPA



HTML & CSS

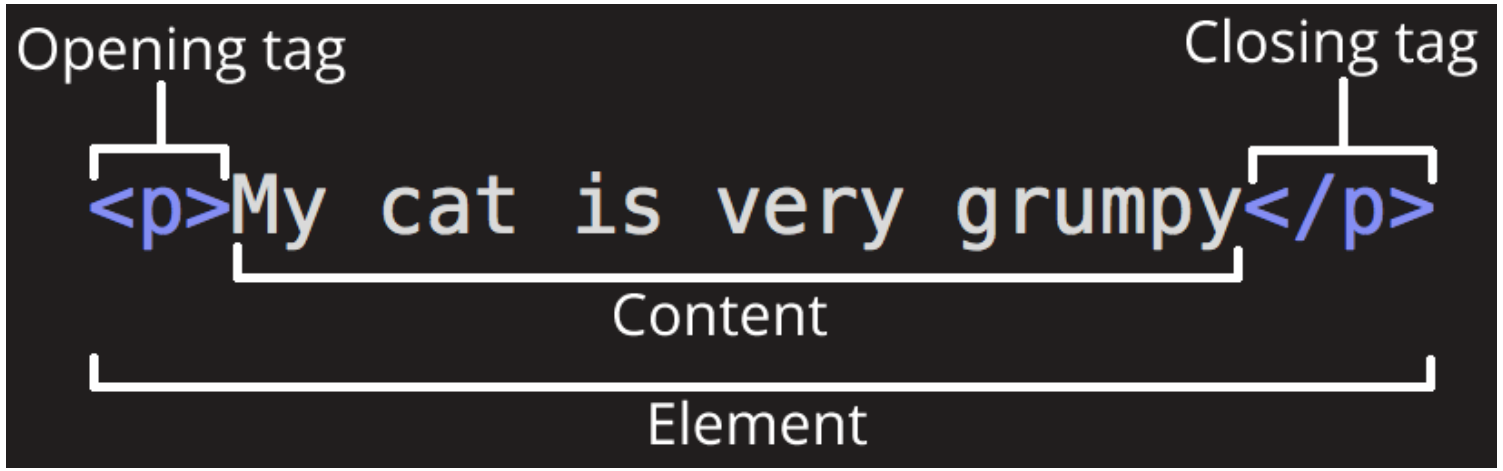
HTML

- HyperText Markup Language (HTML) is the default language of the Web
  - Part of the original Web proposal
  - Derived (also) from SGML
- Markup: used to annotate (hyper)text/media
- Interpreted by a browser

Timeline of HTML

Version	Year	
HTML	1991	Initial version by TimBL and Dan Connolly
HTML 2.0	1995	Published as RFC 1866
HTML 3.2	1997	Published as W3C Recommendation
HTML 4.01	1999	Strict/Transitional/Frameset variations of the language – HTML 4.01 Strict is aka ISO/IEC 15445:2000
XHTML	2000	Based on v4.01, reformulated using XML 1.0; abandoned in favor of HTML5 (see also XHTML5.1)
HTML5	2014	Current version v5.3 (January 2021) HTML Living Standard by WHATWG see history note

HTML elements



- Nesting of elements has to be proper i.e. resolving into a tree
- Inline (same line) and block (new line) elements
- Empty elements are possible e.g. `<img …>`
- Explicit identification by means of `id` attribute

HTML Attributes

Attribute

```
<p class="editor-note">My cat is very grumpy</p>
```

- Spacing and proper formatting of attribute is important
  - Quotation marks are dropped sometimes but this is bad practice
  - Single or double quotation marks are interchangeable as long as their use is consistent
- Boolean attributes: only one value allowed, usually the same as the attribute name e.g. `disabled`

Text

- Heading elements `<h1>` to `<h6>` represent hierarchically nested levels of content in the document
- Paragraph element `<p>` contains paragraph text with whitespace/line wrapping ignored
- In-paragraph text can use additional markups
  - `<em>` for *emphasized text*
  - `<strong>` for text with a **strong emphasis**
  - `<sup>` for $^{\text{superscript text}}$
  - `<sub>` for $_{\text{subscript text}}$
  - `<code>` for `code examples`
  - …

Text (cont.)

- `<em>`, `<strong>`, etc. have semantics defined by the language itself
  - Enforced by the browser
  - Understood by search engines, third-party viewers like screen readers, etc.

- Up to HTML5 presentational elements like `<b>` (bold), `<i>` (italics), and `<u>` (underlined) were also allowed but without semantics i.e. result may vary
  - HTML5 gave them semantic roles, but they should be avoided for accessibility purposes

Lists

- Similar to LaTeX structures
- Basic flavors
  - Unordered: list item elements `<li>` nested under element `<ul>` (unordered list)
  - Ordered: as above but under `<ol>` element
- Description lists also available
  - Pairs of `<dt>` (description term) and `<dd>` (description definition) elements under `<dl>` element
- List nesting is allowed as long at it is proper

Links

- Inline link through `<a>` element with `href` attribute containing the target URI
  - `title` attribute can be used for additional information (as hovertext in browsers)
  - Relative URIs are encouraged were possible

- `<a>` also allows for block level links e.g. around images

- Links to specific part of HTML document are possible by appending the (relevant) URI with # and the fragment marker defined as an `id` attribute to target element
  - e.g. `<a href="index.html#top">` to `<h1 id="top">`…

Images

- `<img>` for images with attributes `src` (source URI) and `alt` (alternative text description)
  - `title` attribute for further information (not recommended)
  - Size control through `width` and `height` attributes
  - HTML5: Attributes `srcset` and `sizes` allow for resolution switching depending on device, plus semantic wrapper `<picture>` for more control
  - HTML5 provides also `<figure>` and `<figcaption>` as a semantic container for figures and their captions, respectively

Other media

- Up to HTML5 embedded video/audio handled by non-native Web technologies (e.g. Flash)

- HTML5 defines native mechanisms for media
  - `<video>` and `<audio>` elements
  - Either use default `controls` attribute or build a JavaScript API instead
  - Nested `<p>` element to be used as fallback content

  - Note: effectively breaks compliance to SGML because they enforce control instead of markup

Tables

- `<table>` represents a table as a sequence of rows
  - `<tr>` represents a row as a sequence of cells
  - `<td>` encloses the contents of a cell
  - `<th>` contains header data
  - `<rowspan>` and `<colspan>` to be used for cells spanning multiple rows and columns, respectively
  - HTML5 allows for separation between header/footer elements under `<thead>`/`<tfoot>` and body elements under `<tbody>`

- Not to be used for substituting the layout of a page
  - Reduces accessibility
  - Results into tag soup through ever more complex markup
  - Not automatically responsive (sized according to content, not overall page area)

Forms

- `<form>` defines a form as an interaction point with a user as a composition of one or more widgets
  - Native widgets include text fields, buttons, etc.
  - Can be further structured through headers and sections
  - Requires processing on the server side
- Server URI to process the data is defined by `action` attribute (default: same URI)
- HTTP method to be used defined by method attribute (default `GET`)
  - `method= "get"` appends data as query parameters
  - `method= "post"` appends data to the HTTP request body

Forms structure

```
<h2>Contact information</h2>
  <fieldset>
    <legend>Title</legend>
    <ul>
      <li> <label for="title_1"> <input type="radio"
id="title_1" name="title" value="Mr."> Mister </label> </li>
      <li> <label for="title_2"> <input type="radio"
id="title_2" name="title" value="Ms."> Miss </label> </li>
    </ul>
  </fieldset>
```

- Grouping through `<fieldset>` element, preferably described by a `<legend>`
- `<label>` describes the `<input>` element
- `<input>` defines the `<type>` and `<value>` of the form element

Form validation

- Data passed from the client to the server are in principle not to be trusted → some validation required

- Types of validation:
  - Client-side, i.e. in the browser before submitting
    - Through JavaScript
    - By means of built-in form validation as of HTML5
  - Server-side, i.e. through application logic
    - Less user friendly/reactive
    - Supported by the vast majority of server-side frameworks

HTML(5) document structure

Sidenote: wrappers

- `<span>` and `<div>` are inline and block, respectively, *non-semantic* elements
  - They should be used only if no other element is suitable
  - Or if no specific meaning is to be attached
  - In practice used with CSS for decoration of text

- Beware of the code clutter

HTML document structure (cont.)

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8"/>
    <title>My page title</title>
    <link
href="https://fonts.googleapis.com/css?family=
Open+Sans+Condensed:300|Sonsie+One"
rel="stylesheet" type="text/css"/>
    <link rel="stylesheet" href="style.css"/>
  </head>

  <body>
        <!-- Here is our main header that is
used across all the pages of our website -->
    <header>
      <h1>Header</h1>
    </header>

…
```

- The `<head>` element contains metadata for the document
  - `<title>` for the title of the document
  - `<link>` to an external stylesheet or `<style>`
  - `<meta>` for other metadata
    - Multiple `<meta>` elements allowed
    - Keywords enable search engine optimization
- Actual content is nested under the `<body>` element

HTML document structure (cont.)

```
...

<nav>
  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">Our team</a></li>
    <li><a href="#">Projects</a></li>
    <li><a href="#">Contact</a></li>
  </ul>

<!-- A Search form is another commmon non-linear way
to navigate through a website. -->
  <form> <input type="search" name="q"
  placeholder="Search query">
    <input type="submit" value="Go!">
  </form>

</nav>

<!-- Here is our page's main content -->
<main>
<!-- It contains an article -->
  <article>
    <h2>Article heading</h2>
    <p>Lorem ipsum dolor sit amet, ...

...
```

- `<nav>` provides navigation inside the inside
  - Outgoing links should be avoided
- `<form>` is used here to navigate
- Actual content goes in `<main>`

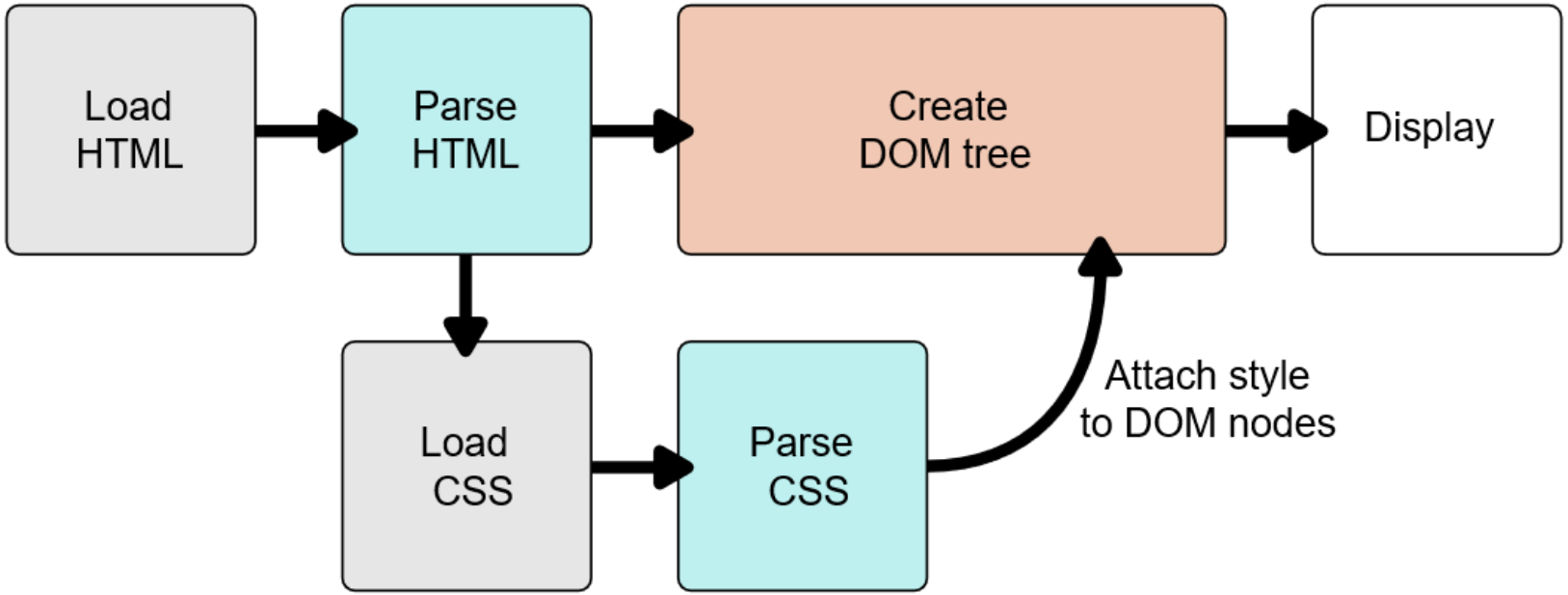
CSS

- Cascading StyleSheets (CSS) specifies how documents are presented to users e.g. styling, lay out etc.
  - Document = text structured using a markup language
  - Presentation = conversion into a usable form for an audience, e.g. done by a browser

- A stylesheet is a collection of CSS rules

- CSS rules are applied to a HTML document to define how it is displayed
  - Properties: values set to update HTML content for display
  - Selector: which elements to apply the property values to

HTML rendering by browsers



- Document Object Model (DOM) represents the document structure as a tree-like structure
- Browser displays DOM contents
- Browsers come with a default style sheet to apply to all pages without one

Selectors

› Define target HTML elements for styling (in groups)

```
p {
  color: red;
}
```

› Different types

- Simple (as above)
- Based on attributes/attribute values
- Pseudo-classes of elements in a certain state, e.g. hovered over by pointer
- Pseudo-elements as group of content elements in a certain position with respect to a given element, e.g. first word of each paragraph
- Combinators and multiple selectors

External stylesheets

```html
<html>
  <head>
    <meta charset="utf-8">
      <title>My CSS experiment</title>
      <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

```css
h1 {
  color: blue;
  background-color: yellow;
  border: 1px solid black;
}
p {
  color: red;
}
```

university of groningen

Internal stylesheets

```
<html>
  <head>
    <meta charset="utf-8">
      <title>My CSS experiment</title>
      <style>
            h1 {
                  color: blue;
                  background-color: yellow;
                  border: 1px solid black;
            }
            p {
                  color: red;
            }
      </style>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>
```

Inline declarations

```
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
  </head>
  <body>
    <h1 style="color: blue;background-color:
yellow;border: 1px solid black;">Hello World!</h1>
    <p style="color:red;">This is my first CSS example</p>
  </body>
</html>
```

- To be avoided except if absolutely necessary

Inheritance and hierarchies

- Inheritance of properties defines if they apply to children elements
  - Some, but not all properties are automatically inherited to children elements
  - Otherwise element gets the `initial` value
  - Can be explicitly controlled by elements such as `inherit` and `initial` (default element style)

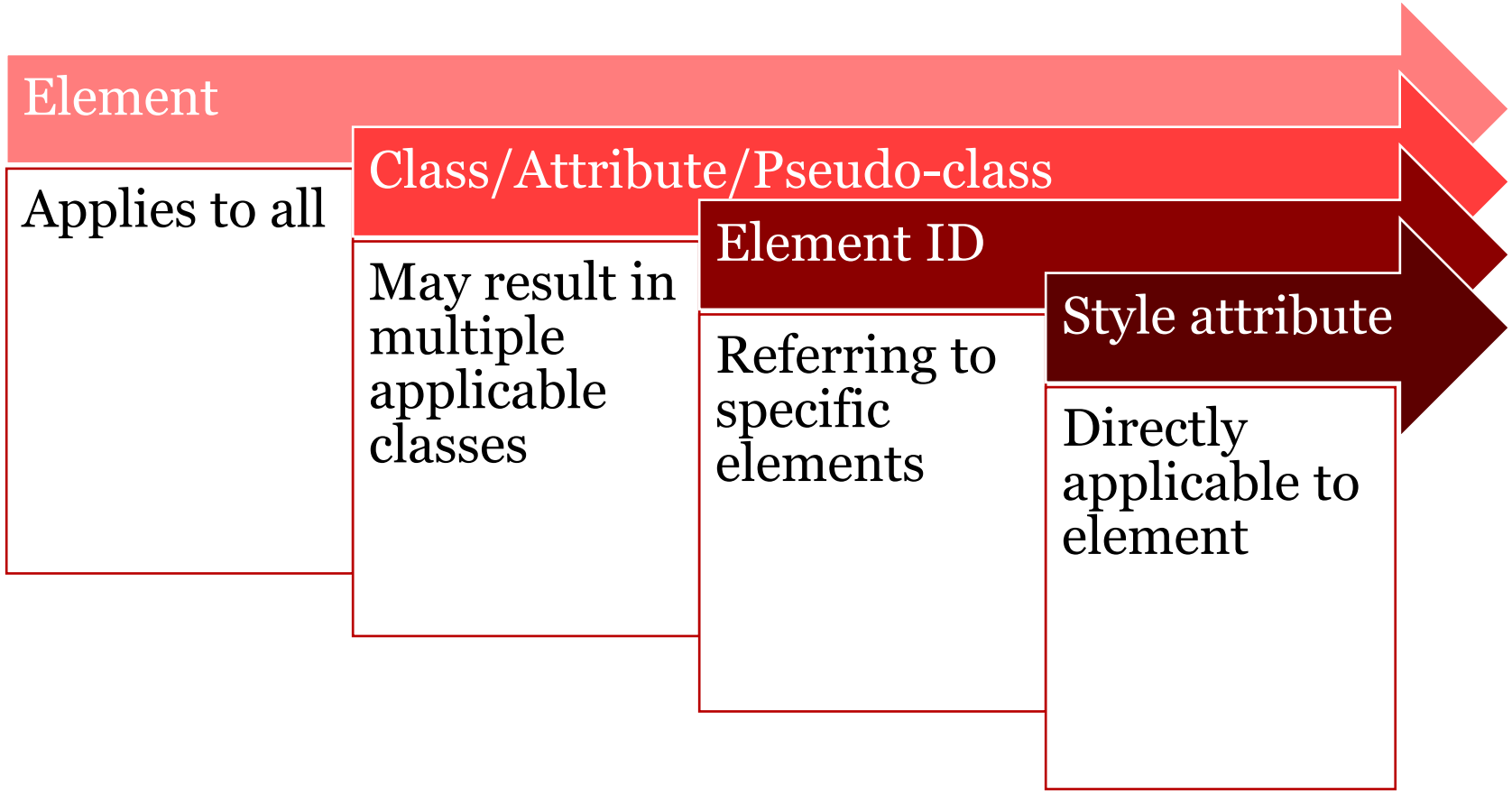
```
p {
  color: green;
  border: medium solid;
}

<p> This paragraph has green <em>emphasized text</em> with a single border around it.</p>
```

Cascade

- The cascade mechanism resolves potentially conflicting rules selecting the same element:
  - Stylesheet origin of the rule is considered:
    author (developer) > user(-defined) > user-agent (browser)
  - Importance (through `!important` attribute) > Specificity (how many elements could be matched) > Source order (later rules in the same sheet win)

- Specificity calculated as a summed weight

CSS Specificity (less to more)



Conflict resolution examples

```
#myElement {
  color: green;   /* green wins*/
}

.bodyClass {
  color: yellow;
}
```

```
#myElement {
  color: green;
}

#myApp [id=“myElement”] {
  color: yellow;   /* yellow
wins*/
}
```

```
p {
  color: green;
}

<p> This paragraph has green text except from the <em
style=“color=purple”>emphasized text</em> that is purple.</p>
```

Static vs Dynamic Web pages

Static Web pages

- Web pages as sets of (HTML + CSS) documents served to client by a (HTTP) server
  - Page content is predefined – known a priori
  - Interaction through e.g. forms is allowed

- Went out of style with the advent/massive adoption of JavaScript (see following) but currently making a comeback
  - Static does not necessarily mean HTML + CSS **only**

Dynamic Web pages

- Enable rich interaction model with the client by supporting dynamic (potentially unknown a priori) content to be served
- Initially started with server-side content generation
- Conversation has shifted a lot to client-side processing in the last years
  - Client = browser or "headless browser" component standing in for the browser

A very compressed history of dynamic Web frameworks (server-side)

- Common Gateway Interfaces (CGIs) were available since the beginning of the Web
  - Allow Web servers to talk to back-end programs and scripts and generate HTML pages in response
  - Programs/scripts in pretty much any language e.g. C (!)
  - Under a `cgi-bin/` folder by convention

- PHP (PHP: Hypertext Preprocessor) started as a set of CGIs in C extended with web form management and database communication (in 1995)
  - Currently in version 8.0
  - Usually but not necessarily comes as a LAMP stack (Linux, Apache [Web Server], MySQL, PHP) package

A very compressed history of dynamic Web frameworks (server-side)

- Java servlets and JavaServer Pages (JSP) – 1996 and 1999 respectively
  - Servlets are software components written for a server (e.g. Java EE Server) but end(ed) up being used to implement Web containers by embedding HTML in Java code
  - JSPs generate pages similarly to PHP by embedding Java code in HTML

- ASP.NET (Active Server Pages .NET) – 2002
  - Basically the equivalent of PHP and JSP for Microsoft's .NET framework

A very compressed history of dynamic Web frameworks (server-side)

- Django (2003)
  - Python-based MVC/MVT (Template) framework
- Flask (2004)
  - Python-based micro-framework
  - Started as an April's Fool joke
- Ruby on Rails (2004/5)
  - Rails as a Model View Controller (MVC) framework for database-backed web applications
  - Written in Ruby
  - Influenced a number of other frameworks

A very compressed history of dynamic Web frameworks (server-side)

- Express(.js) – 2010
  - Built around the Node.js runtime environment for server-side JS execution
  - Comes usually but not necessarily in the MEAN stack (MongoDB, Express, Angular, Node.js)

- Laravel (2011)
  - MVC-oriented PHP framework

- To be extended…

Client-side dynamic Web pages

- Supported as of HTML 4.0 through `<script>` element

- JavaScript as a high-level scripting language
  - To be executed in the browser like Java Applets, VBScript programs, or ActiveX controls in pages
  - Apparently both the parent and the child of ECMAScript aka ECMA-262 specification for general purpose scripting languages
  - Also in a complicated relationship with TypeScript which defines strong types of JavaScript

AJAX

- Asynchronous JavaScript and XML for client-side applications
  - Non-blocking interaction with the back-end
  - XML not strictly necessary

- Combines multiple technologies:
  - HTML & CSS
  - DOM
    - Note: DOM page model ≠ HTML page model ≠ XML model
  - XML to let clients exchange application data with the server – these days mostly JSON
  - JavaScript as the language that binds everything together

AJAX

- Allows clients (browsers) to:
  - Make requests without reloading the whole page
  - Receive and process data from the server

- Builds on XMLHTTPRequest (XHR) objects
  - Can retrieve any type of data, not only XML
  - Can retrieve data both synchronously and asynchronously
  - Supports listening to events, including server progress
  - Responses are DOM objects representing XML documents

jQuery x AJAX

- jQuery as a JavaScript library offering methods for common tasks
  - Document traversal and manipulation
  - Event handling
  - Animation

- Simplifies using AJAX by offering library of AJAX-specific methods

Fetch API

- Interface for fetching resources (to be implemented by the browser)
- Manages `Request` and `Response` objects

```
const request = new Request('https://example.com', {method: 'POST', body:
'{"foo": "bar"}'});

const url = request.url;
const method = request.method;
const credentials = request.credentials;
const bodyUsed = request.bodyUsed;
```

Fetch API

- Actually fetching a resource through the `fetch()` method
- Returns a `Promise` object that eventually resolves into a `Response`

```
fetch(request)
  .then((response) => {
    if (response.status === 200) {
      return response.json(); // Actually a second promise to parse the response
body into JSON
    } else {
      throw new Error('Something went wrong on API server!');
    }
  });
```

Single Page Applications

- Traditional, multi-page Web applications (MPAs) minimized client-side behavior

- Single Page Applications (SPAs) are heavily client-side dynamic Web pages
  - Content is loaded from a static HTML page + JavaScript libraries to run the application
  - All interaction in one page

MPA vs SPA

Factor	Multi-Page Applications	Single Page Application
Required Team Familiarity with JavaScript/TypeScript	Minimal	Required
Support Browsers without Scripting	Supported	Not Supported
Minimal Client-Side Application Behavior	Well-Suited	Overkill
Rich, Complex User Interface Requirements	Limited	Well-Suited

Table source

- Acceptable/common to combine both depending on what is to be delivered

Web Frameworks for SPAs (client-side)

- Angular
  - Led/developed by Google
  - Builds on TypeScript as of Angular 2
  - Component-based model (basically MVC)
  - HTML templates assigned to components
- React
  - Led by Facebook/Instagram
  - Actually a library (not a framework)
  - Allows for HTML injection to JS code
- Vue
  - Borrows heavily from Angular but is less heavy/complicated
- See also this comparison article

Server-side rendering (SSR) for SPAs

- Rendering the whole application server-side
  - Returning HTML to the client

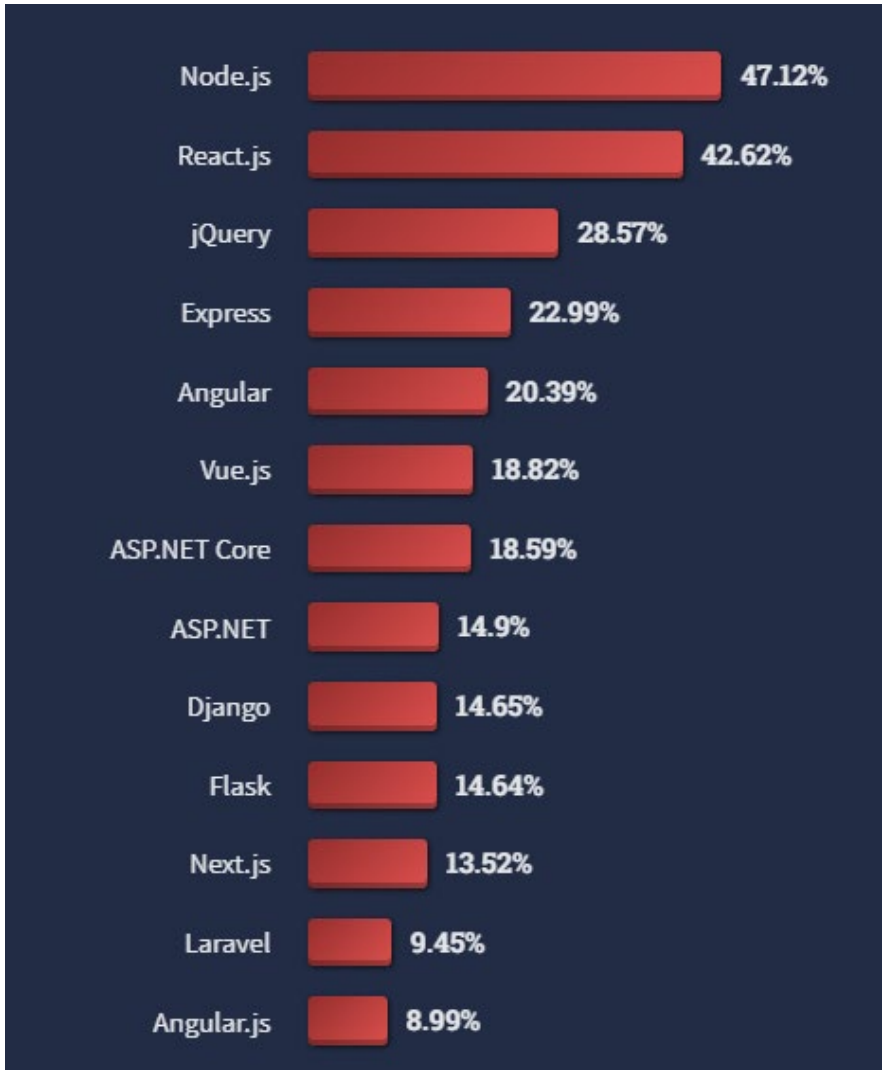
- Client-side Web frameworks also support SSR
  - Angular Universal for Angular
  - Next.js for React
  - Nuxt.js for Vue

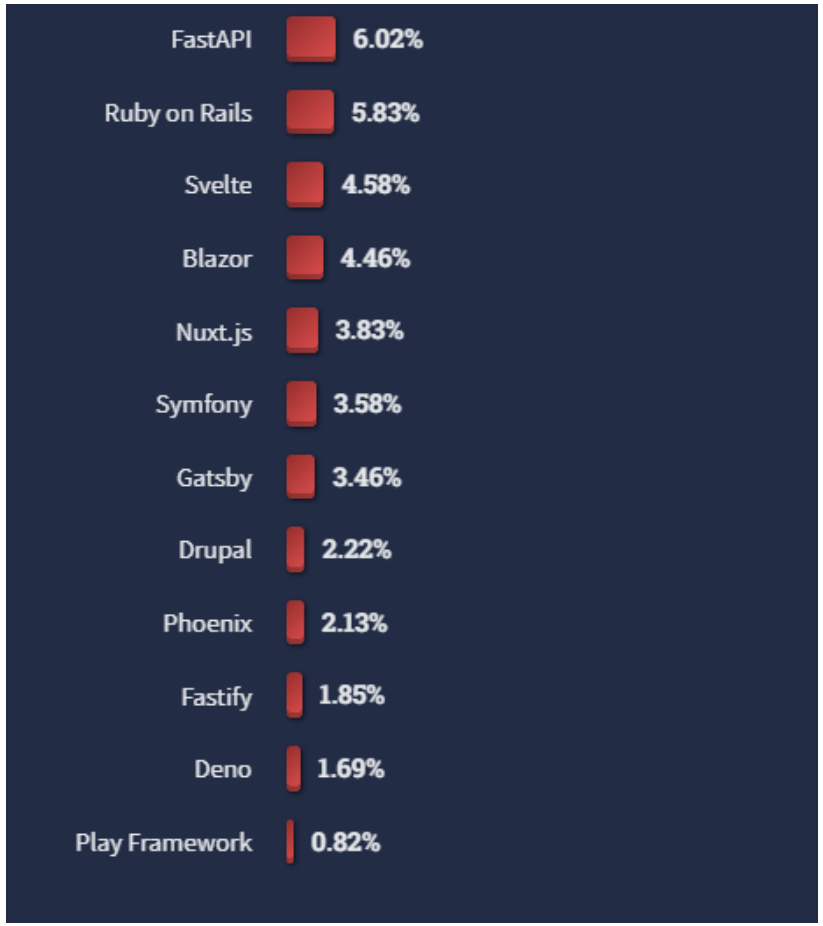
Other server-side options

- Content Management Systems (CMS) such as Wordpress
  - Allow users to create content without coding
  - Require time and knowledge to setup
  - Impose limitations
  - Available as downloadable software or as a service

- Static site generators, e.g. Jekyll
  - Dynamically generate webpages of MPA apps server-side
  - May include JavaScript but rendered by the server
  - Have their own learning curve

Popularity of Web (server/client-side) frameworks





Source: stackoverflow annual survey 2022
See also: popular tech stacks on stackshare.io

Source material

- MDN's web development guide

Supplementary material

- Web Design in 4 minutes by Jeremy Thomas
- Web technology for developers at MDN
- Tutorials at w3schools

Self-evaluation questions

- What is the difference between text elements with semantics and those without in HTML? Which ones are recommended to be used, and why?
- How was embedded video and audio handled up to HTML5? How are they handled by HTML5? What are the implications of this mechanism?
- What are the types of HTML form validation available, and when are they to be used?

Self-evaluation questions (cont.)

- How is the cascade mechanism used to resolve conflicts between CSS rules?
- What is the order of specificity in for CSS rules?
- Under which conditions would you recommend the use of a SPA-style application?
- What options are available for creating server-side dynamic Web pages?

Next lecture(s)

[Tutorials]