Sofascore Frontend Academy Lecture 01,
February, 2024

Frontend. Browser. HTML. CSS.

Alen Murtić

Sofascore

TABLE OF CONTENTS

01	About Sofascore academy and your teachers
02	Web applications and browser
03	HTML
04	CSS + Appendix

Sofascore

01

About Sofascore academy and your teachers

Sofascore academy

- Sofascore started student education relatively early
  - Some of the first employees were students
  - Student courses with 3-4 lessons from late 2015 to 2017
  - Summer internships in 2018 and 2019
  - Sofascore academy in this format from 2020 (this is 5th edition)
    - Great for sharing knowledge, expanding community and potentially expanding Sofascore team

Your Academy Manager - Alen Murtić

- Started student job at Sofascore in 2017
    - Backend developer with potential switch to data analytics (did not switch :D)
- Lead Symfony portion of Sofascore backend academy in 2020 & 2021
- Switched teams to frontend in 1/2021, attended 2021 frontend academy
- Lead 2022, 2023 and 2024 frontend academy

Teachers - Darjan Baričević and Petar Ćorluka

- Frontend devs working at Sofascore since 2022
- Did not attend academy (losers)
- Reviewers for the most homeworks
- Each of them will cover 50% of the lessons

Assistant - Tvrtko Babić

- Former academy graduate (2022), now working full-time at Sofascore
- Reviewer for couple of homeworks

Planned curriculum

1. Frontend. Browser. HTML. CSS. ||| Hw: Init git repo and solve CSS quiz.
2. Responsive web. JavaScript. ||| Hw: Solve various JS tests.
3. Typescript. Promises. Fetching data. Event propagation. ||| Hw: Simple web app || I-O script
4. React.js ||| Hw: Some kind of simple React app.
5. CSS in JS (Panda CSS). Next.js basics. ||| Hw: Review previous homework.
6. More of Next.js. Zeplin/Figma. ||| Hw: Init final project, one project page
7. Context. Router. More hooks. ||| Hw: Review previous homework.
8. SWR. Client vs server. Routing. ||| Hw: complete final project with one checkpoint.
9. RSC and app folder in Next.js
10. Redux - optional lesson

Curriculum changes from last year

- CSS-in-JS: now more styled-components, we will use Panda CSS now
- Next.js: Pages router is a thing of past, we will cover App router now

02

Web applications and browser

Web application

- Wikipedia: link
    - "A web application (or web app) is application software that runs on a web server"
    - "Web applications are accessed by the user through a web browser with an active network connection"
    - "These applications are programmed using a client–server modeled structure"
- Simply: it consists of **Frontend (client)** and **Backend (server)**

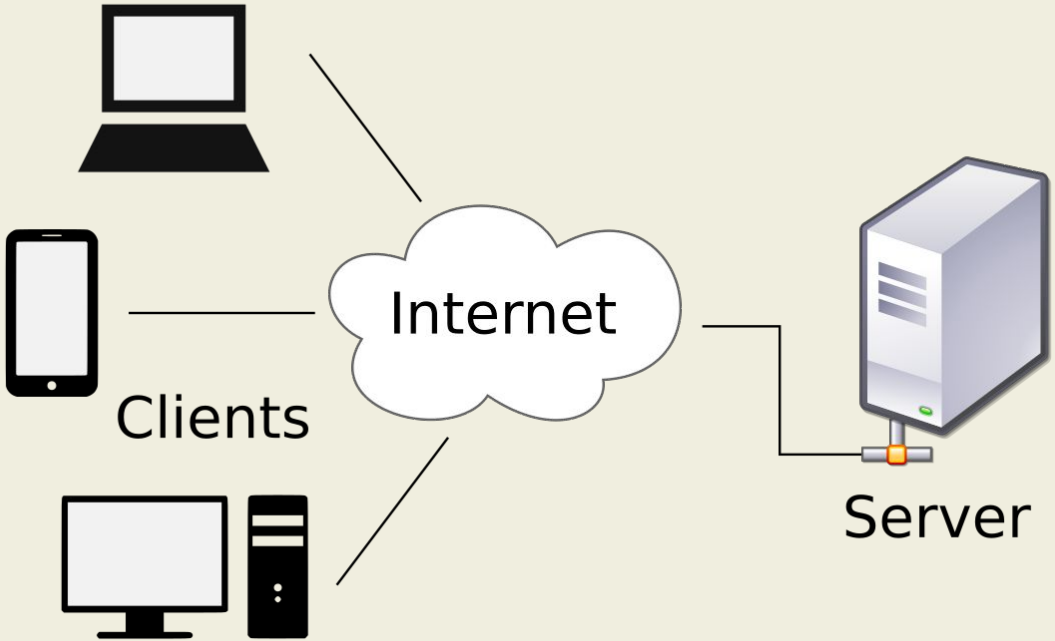
Client - Server Architecture

- Core principle of web communication
    - Client asks the server for a resource, server responds
    - Resource: HTML document, formatted data, image, video,...

  E.g. What is the score in a match? Give me team logo...
- THE protocol: **HTTP** (Hypertext Transfer Protocol)
    - What is HTTP (Cloudflare)?
    - Later created: Websockets - to improve efficiency

Client - Server architecture

- Basically: clients pull data from server or push it to server via Internet
- Communication protocol: HTTP



Frontend

- Interface with which a user (person or a script) interacts (sees, clicks, ...)
  - "Visible" part of the web application
  - In a general meaning - any client which has UI - Android, iOS, KaiOS apps
  - We use the term "Frontend" as a shorthand for **web frontend**

Web frontend

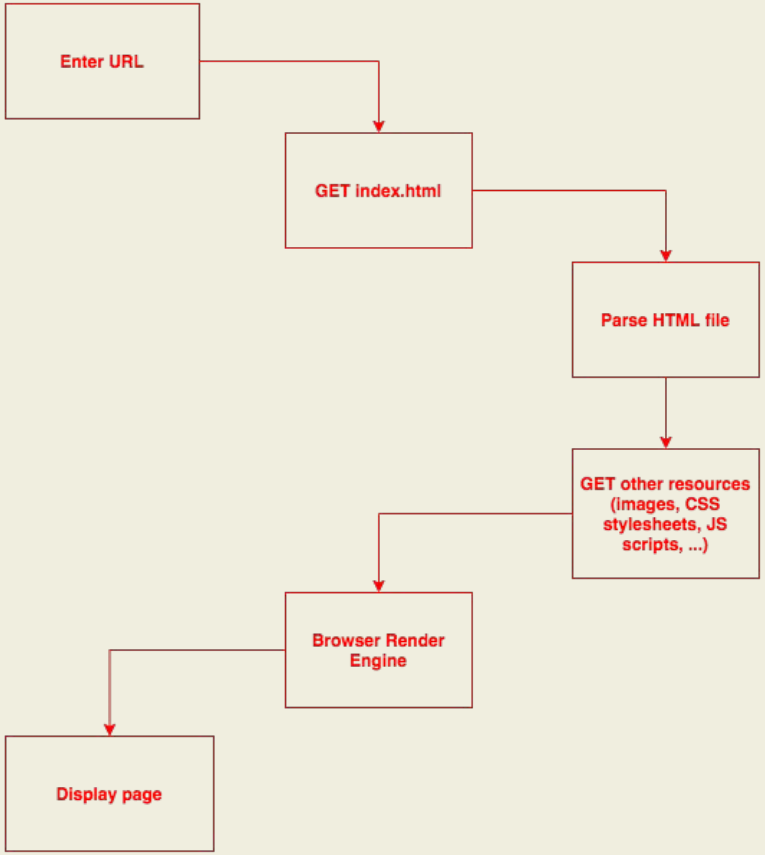
- Visual application that is displayed by web browser
- Source written in HTML, CSS, JavaScript
- Source can be in WebAssembly 😱 - Binary (compiled) [mostly JavaScript] for higher performance
    - Writing WebAssembly By Hand
    - Personal opinion: not a fan of writing code directly in WASM

Browser

- A tool to navigate the web, display web applications and provide interactivity
- Core component: render engine
  - MSHTML, EdgeHTML - deprecated, used by Internet Explorer and early Edge
  - Gecko - Firefox
  - WebKit - Safari and early Chrome
  - Blink - Chromium project, fork of WebKit
  - Chrome, current Edge, Brave, Opera, Vivaldi, Samsung Internet, ...
- 2023: For desktop I would recommend Firefox -> reason: Manifest V3
  - Link 1 and link 2 why, Vivaldi and Brave try to fight it, Edge doesn't
- For mobile it's a little bit murkier, but the Android Firefox is getting better, on iOS everything is repackaged Safari (for now)

Browser flow

- Example of client - server architecture
- Browser (+user) = client
- Detailed description: how browsers work



Browser differences

- Different engines -> differences in how everything works
  - Most of things are standardized via W3C, but some browsers don't support some features
  - Can I use "navigator.share"?
  - Browsers depend on the OS for features like graphics APIs, threads, processes, ...
- Browsers work on CPU, but do mostly graphics tasks
  - Hardware acceleration -> doing some tasks on GPU (or special CPU cores)
  - Problem with HA: now your browser depends on your GPU & its driver

Reporting bugs cheatsheet

- Always report: browser, OS
- Can make a difference: ad-blocker, tracking protection, private (incognito) or normal mode
    - Also if 3rd party cookies are enabled or not
- Nice caveat: hardware acceleration
    - e.g. Chromium browsers and image scaling with hardware acceleration in versions 80-sth

03

HTML

HTML

- **H**yper**T**ext **M**arkup **L**anguage
  - HyperText -> text with references to other pages (links)
  - Markup -> standardized set of notations (tags and attributes)
    - e.g. XML, markdown (.md), TeX/LaTeX
  - Idea: How to display content
  - NOT A PROGRAMMING LANGUAGE!!!! MARKUP LANGUAGE!!!

HTML

- Created by Tim Berners-Lee to enable document sharing (text based -> links, headings, paragraphs)
  - Standardized by W3C
  - Latest and greatest: HTML5 - late 2000s, big improvements
    - Made proprietary things such as Flash obsolete
- HTML6 is coming - 4 years old article 😂
- HTML is forwards-compatible
  - Designed to treat all tags in the same way (as inert, unstyled inline elements) unless their appearance or behavior is overridden
  - i.e. 2007 browser can display 2023 plain-HTML page decently

HTML structure

- **HTML Element: Tag + Attribute(s) + Content**
- Tag: identifies element (html, body, b, div, span)
  - opening: e.g.
  - closing:
  - self closing: <img/>
- Attribute: specifies properties of an element (e.g. styling, source for an image, ...)
- Content: between opening and closing tag
  - any text, HTML element, ...
  - self closing tags don't have content
- Each HTML document has **html**, **body**, **head** tags.

HTML

HTML elements examples

- `<b>This is bold text</b>`
- `<div id="atribute_example">Text and/or other element(s)</div>`
- `<img src="path_to_image" alt="Text if image fails to load" />`

HTML sample

- **HTML relations:**
- Parent - Child -> child is parent's content
- Siblings - two elements with the same parent

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Sofascore Frontend Academy</title>
  </head>

  <body>
    <p>Hello!</p>
  </body>
</html>
```

Semantic HTML

- The same content can be described in different ways
- Write HTML in a way it conveys meaning when read, not just in browser

HTML examples

04

CSS

CSS

- **C**ascading **S**tyle **S**heets
  - How HTML elements are displayed on device
  - CSS3 specification
- Syntax: cssProperty: valueOfProperty;
  - e.g. color: blue;
- Cascading -> styles cascade (apply to lower levels) if not overridden

CSS Selectors

- CSS defines styles and can be applied to single element, or to the group of elements
- Inline styling - for single element
  - Text
- Style all elements with the same tag
  - h2 { text-transform: uppercase; }
  - useful for resetting default browser styling (e.g. ul or button elements)
- Style all elements with the same class attribute set to className
  - .className { background-color: tomato; }
- Style all elements with the same id attribute
  - #uniqueId { text-align: center; }

CSS Selectors 02

- Selectors can be mixed
  - h2.specialHeading { padding: 8px; }
  - h2 .specialHeading { margin: 16px 8px; }
- Universal selector (*), Grouping selector (div, p { ... })
- **Specificity: Inline style > Id Styling > Class styling > Tag styling**
- Notes:
  - Id attributes should be unique for each element and should appear only once on each page
  - Same element can have multiple classes (e.g.)
  - Adding !important to value of CSS property will override a rule that can't be overridden in any other way
    - Multiple !important values can make CSS extremely confusing

Adding CSS to HTML

- <link rel="stylesheet" type="text/css" href="myStyleSheet.css">
- Embedded in <style></style> element
- Directly on element
- It is applied in order they are linked and order in the file in which they were defined
- Do separate HTML and CSS files!

CSS

CSS Box model

- Each element is a box defined with x, y, width and height
  - x and y mark the top left edge of the box
  - width and height are applied to content part of the box
- Where are content, padding, border and margin on the cats image?

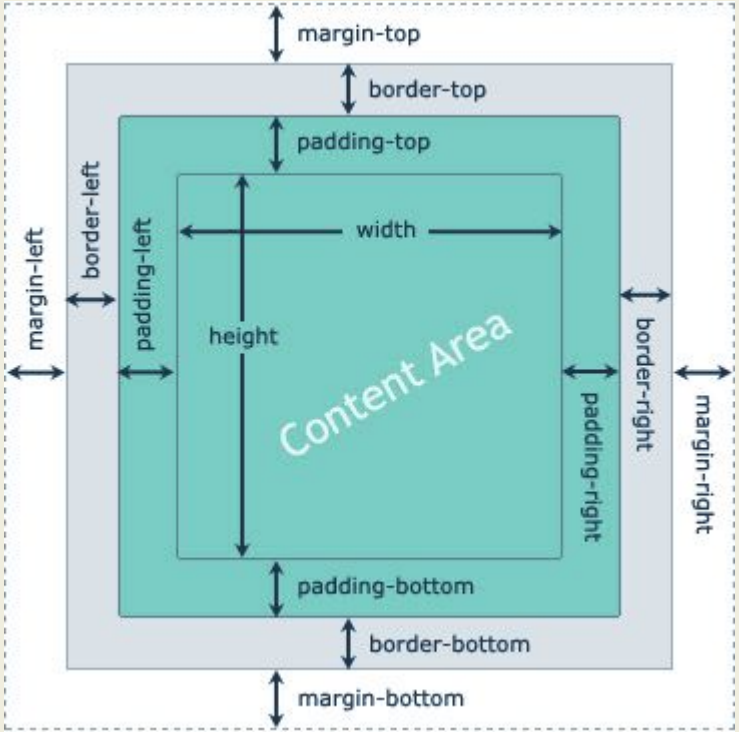
CSS

CSS Box model

- Content can be distributed in the box with padding
- Box can be spaced from other boxes with margin

  (e.g. spacing between sibling elements)

  - Margins of adjacent elements don't stack!!! They collapse. Bigger wins.
  - Rules of margins
  - Margin can be negative - a bit hackish
- Box can be made visible with border - draws line on the edge of the box



CSS Box model and box-sizing

- box-sizing property changes how browser calculates size of the element
  - Whether to border or padding included or excluded from the width
  - Default value is content-box, which includes only content
  - We at Sofascore use box-sizing: border-box -> includes content, padding and border
    - Very simple example why border-box can be superior: link

CSS examples

Thank you for your attention!

Sofascore

CSS Appendix

01

Display

Display

CSS Display Properties

- Display property is used to specify how should element behave in the parent element.
- Elements have default display properties defined in HTML specification
- Common display properties:
  - block
  - inline
  - inline-block
  - none - removes element from screen
  - grid, table, **flex** - special layouts

Display

CSS Display Properties

- Display: block - e.g. div, ul, main, p
  - Element will take as much space as possible in the single row
  - No two block elements in the same row
- Display: inline - e.g. span, b
  - Element will use minimal width needed
  - Ignores width and height set
  - Possible multiple elements in the same row
- Display: inline-block - e.g. img
  - Inline element which doesn't ignore width and height set

02

Flex

Flex

Flexbox - display: flex

- Displays element as block level container - it appears the same, children will be changed
- Applies flexible width (or height) and positioning to children
- Aligns children on defined axis - row, column
- Children positioning can be specified
    - Align all with start, center or end of the parent container
- If children take more space than available - define wrapping
- If children take less space than available - define layout

Flexbox - display: flex

- Best additional resources to learn flex
  - W3C - Schools - usually great, but it’s not their best work, in my humble opinion
  - A Complete Guide to Flexbox - phenomenal, great read even if you know flex
  - Flexbox Froggy game - when you feel ready, try this out
- Flex vs grid
  - Flex can do everything grid can, not exactly visa versa
  - Flex is more supported on older browsers, but it’s becoming moot point in 2023
  - Grid is better for very simple layouts (designed for Windows 8 start menu)

Flexbox - parent (flex container) properties

- Most important parent properties:
  - flex-direction - the direction (axis) of flex-container
    - row (default), row-reverse, column, column-reverse
  - justify-content - strategy of organizing children along the flex axis
  - align-items - strategy of organizing children along the other axis
  - flex-wrap - whether to wrap children in the next row/column when there isn't enough space
  - gap - default space between children on the container
    - Example of problematic Safari property, but less so in 2024 - Can I use link

Flex

Flexbox - child properties

- Most important children properties:
  - flex-grow - should the child grow and in what ratio
  - flex-shrink - should the child shrink and in what ratio
  - flex-basis - basis dimension for grow and shrink calculation
    - Can be used in combination with [max|min]-[width|height]
  - flex - shorthand for flex-grow, flex-shrink and flex-basis
    - e.g. flex: 3 2 50px;
  - align-self & justify-self - ignore how the container aligns/justifies children and set own rule

Flex

Practise makes perfect

Flexbox examples

Sofascore

03

Position and transition

CSS Position

- Type of positioning used for an element
- Used to define how should element be positioned based on its top, bottom, left and right, properties
- Default value: **static**
    - Ignore position properties
- z-index - tells browser how to display overlapping elements (higher number first)
    - default: first element on top

CSS Position - common values (other than static)

- relative -> position element relative to element's normal position (top, left, ...)
  - stays in normal flow, reserves its space
- fixed -> positions element relative to a screen (not content on the screen)
  - stays on the same place when page is scrolled
- absolute -> positions element relative to nearest positioned ancestor
  - positioned ancestor -> any element with position property different then static (default) !!!
  - common use with relative parent
  - removed from normal flow, no space reserved
- sticky -> positioning depends on scroll position, alternates between absolute and fixed
- MDN position documentation

Position and transition

CSS Transition

- Allow gradual/smooth transition between states, e.g. changing color, appearance, etc.
    - There are many many libraries which do more than basic CSS transition
    - Basic CSS is faster 😁
- THE most common problem in basic CSS:
    - Exiting element has to stay in HTML (DOM) until the animation finishes and then disappear
- We use Framer motion when tasked to do something more than basic CSS
    - Going way ahead of ourselves, but AnimatePresence is easy solution for the problem

Position and transition example

Sofascore