SEBA Master: Web Application Engineering

4. Developing Serverless Single-Page Web Applications

Prof. Dr. Florian Matthes, SS21

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

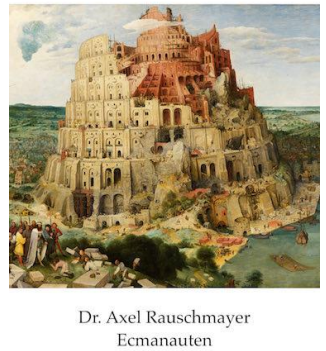
Material and References

https://html.spec.whatwg.org/multipage/

https://developers.google.com/web/

https://developer.mozilla.org

https://javascript.info/



http://exploringjs.com/index.html

Outline

4. Developing Serverless Single-Page Web Applications

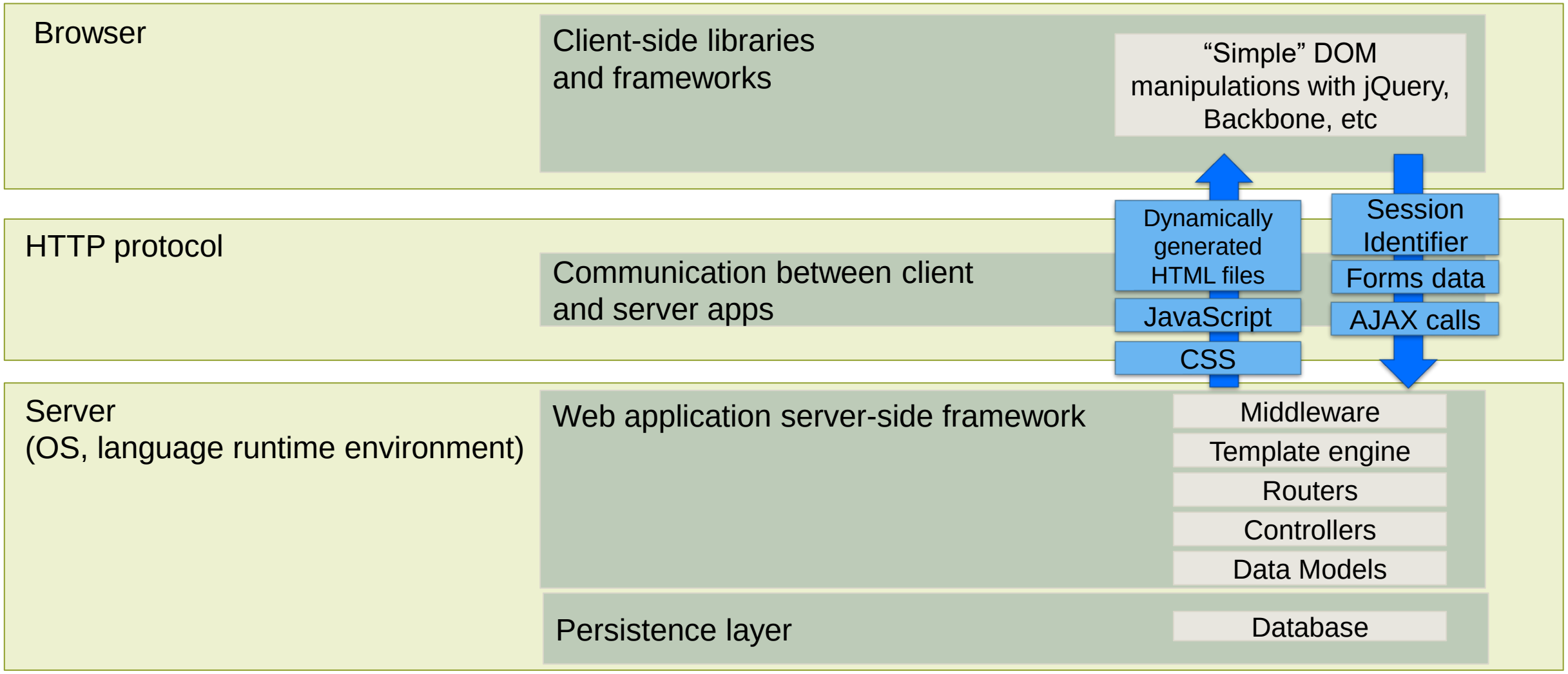
- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Architecture of Web Applications

Client (Web browser)	Client-side libraries and frameworks	• Information presentation • Information retrieval • Information navigation • [Information validation & editing/collaborative editing]
HTTP protocol	Communication between client and server apps	• Established common protocol
Server (OS, language runtime environment)	Web application server-side framework	• Business logic
	Persistence layer	• Persistency management

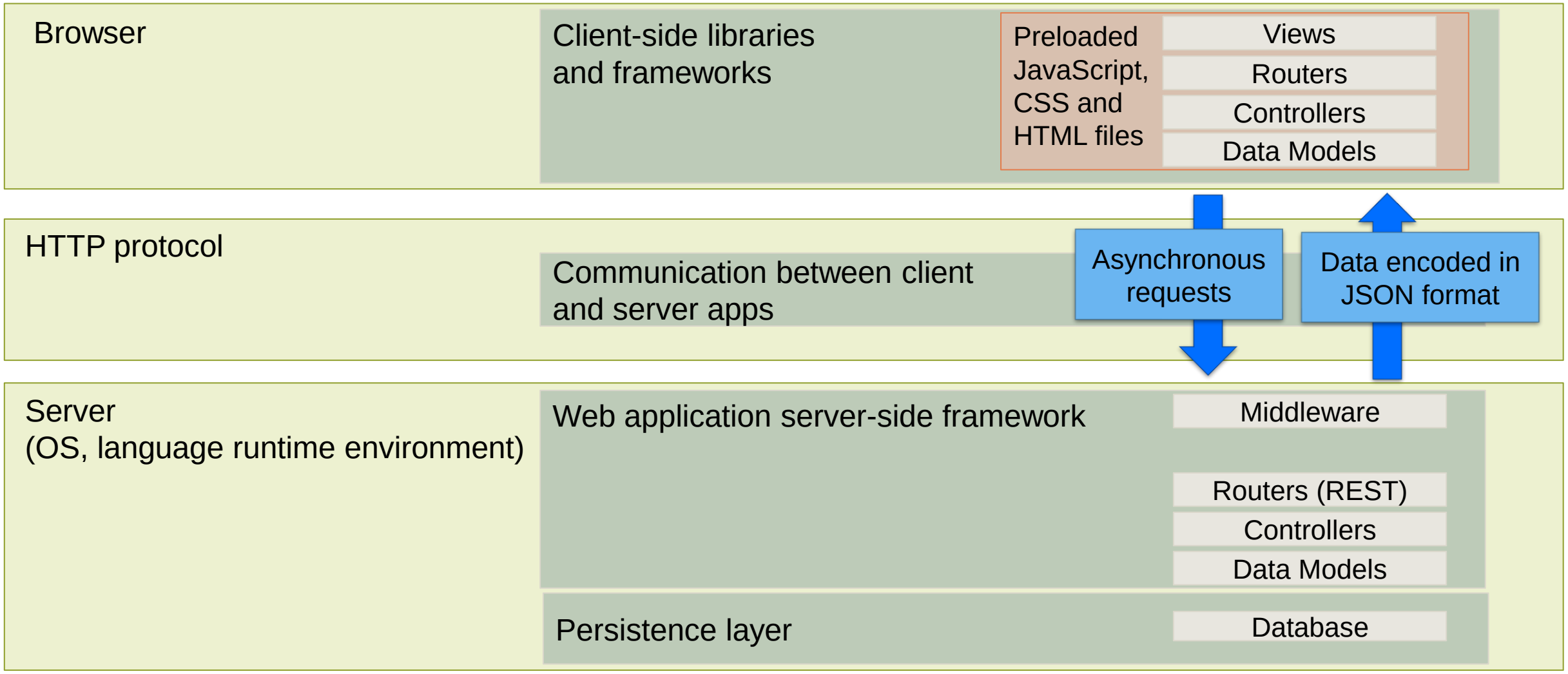
Architecture of Web Applications

Server-side rendering

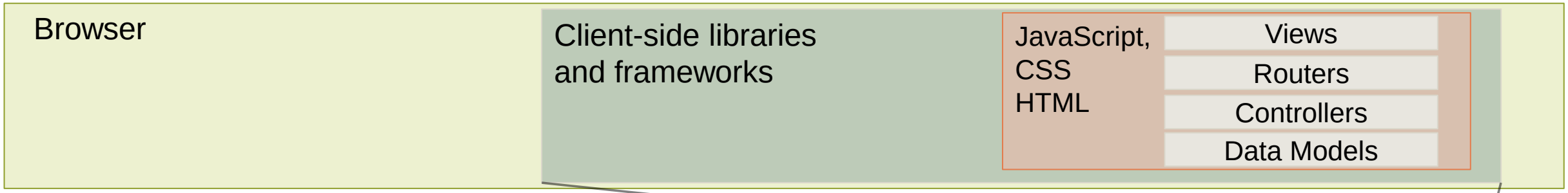


Architecture of Web Applications

Client-side rendering



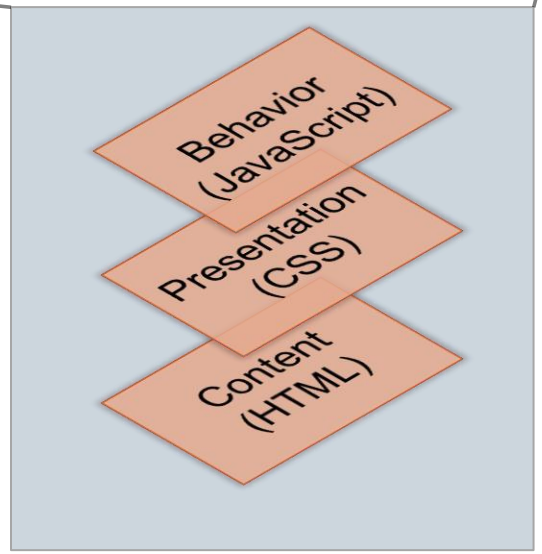
Client-side web applications can be divided in three layers



Content layer
- Is always present
- Is specified using HTML

Presentation layer
- Defines how the content will appear to a human being who accesses the document in one way or another (via a web browser, on a printer, ...)
- Is realized using CSS

Behavior layer
- Involves real-time user interaction with the document
- Is realized using JavaScript



Keeping these three layers separately
- Enables separation of concerns
- Increases the maintainability and understandability.

(http://reference.sitepoint.com/css/css)

Client-side web applications changed a lot though

- Strict separation into HTML, JavaScript and CSS is not common anymore
- In React for instance, different options exist
- One possible setup consists of a single root html file, whereas anything remaining is defined in JavaScript files
  - **JavaScript**
  - **HTML → JSX**
  - **CSS → StyledComponents**

```
"use strict";

import React from 'react';
import Styled from 'styled-components';


class PlainFooter extends React.Component {

    constructor(props) {
        super(props);
    }

    render() {
        return (
            <div className={this.props.className}>
                <hr/>
                <p>© {new Date().getFullYear()} sebis. All rights reserved.</p>
            </div>
        );
    }
}

export const Footer = Styled(PlainFooter)`
    max-height: 35px;
    bottom: 0;
    left: 0;
    right: 0;
    position: fixed;
    background: white;
    > p {
        text-align: center;
        margin-top: 4px;
    }
`;
```

Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Overview of HTML

- HTML (HyperText Markup Language) is designed to specify the logical organization of a document, with important hypertext extensions.
- HTML instructions divide the text of a document into blocks called *elements*.
- Each HTML document consists of two parts:
  - The *body* of the document is to be displayed by the browser.
  - The *head* defines information “about” the document, such as the title or relationships to other documents.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<!--[if lt IE 9]>
  <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
<![endif]-->
</head>
<body>

<section>

<h1>Famous Cities</h1>
<article>
<h2>Munich</h2>
<p>Munich is the capital and largest city of the German state of Bavaria, on the banks of River
Isar north of the Bavarian Alps. Munich is the third largest city in Germany.</p>
</article>

<article>
<h2>Paris</h2>
<p>Paris is the capital and most populous city of France.</p>
</article>

<article>
<h2>Tokyo</h2>
<p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area,
 and the most populous metropolitan area in the world.</p>
</article>

</section>

</body>
</html>
```

Most important HTML tags

- Document structure
  <html>, <head>, <body>, <title>, <meta>, <link>, <script>, <style>
- Basic semantics
  <h1> to <h6>, <p>, , , <blockquote>, <cite>, <ins>, <del>
- Hypertext
  <a>
- Lists
  **<ul>, <ol>, **, <dl>, <dt>, <dd>
- Page structure
  ** , **, <pre>
- Visual formatting
  **
**, <b>, <i>
- Forms
  <form>, <input>, <label>, <textarea>, <button>, <select>, <option>, <fieldset>, <legend>,
- Objects, multimedia, etc.
  <img>, <object>
- Tables
  <table>, <tr>, <th>, <td>, <thead>, <tbody>, <tfoot>

Changes and improvements in HTML5 (2014) compared to HTML4 (1998)

- One important goal of HTML5 is to allow for the development of *rich internet applications (RIA)* without having to rely on proprietary technologies like Adobe Flash or Microsoft Silverlight.
- New semantic elements are added, e.g., **<nav>** (for navigation) or **<footer>** mark certain parts of a web site and can be interpreted by search engines. They define no special behavior.
- Some HTML4-elements used only for formatting were dropped. Developers are encouraged to use CSS instead.
- Multimedia contents can be presented by using tags like **<audio>** or **<video>.**
- A canvas element for 2D-drawings is introduced.
- New form controls for dates, times, email, search inputs and URLs.
- New APIs to support media playback, document editing, offline applications, …

Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Overview

- CSS is a style sheet language used to describe the ***presentation*** (**colors, fonts, layout**) of a document written in a markup language.

- Style sheets can be inserted as
  - *external* styles,

```
<head>
    <link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

mystyle.css

```
hr { color:#A0522D; }
p { margin-left:20px; }
body { background-image:url(images/back40.gif); }
```

  - *internal* styles, and

```
<style type="text/css">
  hr {color: #A0522D;}
  p {margin-left: 20px;}
  body {background-image: url(images/back40.gif);}
</style>
```

  - *inline* styles.

```
<p style="color: sienna; margin-left: 20px">
  This is a paragraph
</p>
```

- History of CSS:
  - CSS Level 1; December 1996
  - CSS Level 2; May 1998
  - CSS Level 3; is divided into separate modules, e.g., “CSS Backgrounds and Borders” and “CSS Multi-column Layout”

Syntax of CSS

- A style sheet consists of ***rules***.
- Each rule consists of one or more ***selectors*** and a ***declaration block***.
- A declaration-block consists of a list of semicolon-separated ***declarations*** in curly braces.
- Each declaration itself consists of a ***property***, a colon, a ***value***, then a semicolon

```
p { color:#000; }
a { text-decoration:underline; }
```

- Properties fall into these categories:
  - **Text** (font-family, font-size, color, text-align, ...)
  - **Background** (background-color, background-image, ...)
  - **Box model** (width, height, top, right, margin, padding, border, ...)
  - **Layout** (position, display, visibility, z-index, float, ...)
  - Other (cursor, list-style-image, ...)

CSS selectors (1)

Selectors are one of the most important aspects of CSS as they are used to "select“ elements on an HTML page so that they can be styled.

- **Type Selectors**
- **Class selectors**

```
em { color:blue; }
```

```
.big { font-size:100%; font-weight:700; }
```

```
<p class="big">This is some <em>text</em></p>
<p>This is some text</p>
<ul id="navigation">
   <li class="big">List item</li>
   <li>List item</li>
   <li>List <em>item</em></li>
</ul>
```

Combining type and class selectors

```
.big { font-size:110%; }
p.big { font-weight:700; }
```

- **ID selectors**

```
#navigation { width:12em; color:red; }
```

This is some *text*

This is some text

- **List item**
- List item
- List *item*

CSS selectors (2)

- **Pseudo-classes**: formatting elements based on their current state

```
a:link,a:visited { color:blue; }
a:hover,a:active { color:red; }
```

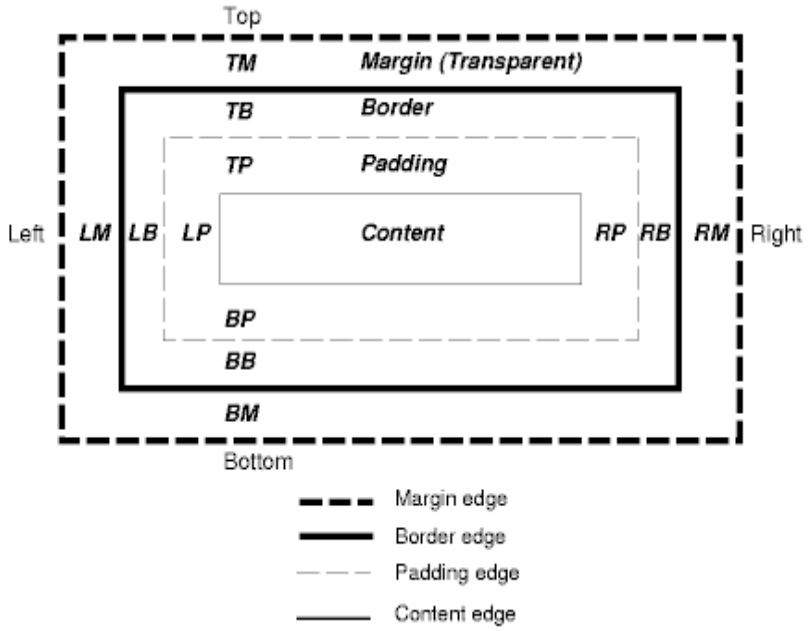
- **Universal selector**: selects all elements

```
* { color:blue;}
```

- **Advanced selectors** (are not supported by IE up to version 6)
  - Child selectors
  - Adjacent sibling selectors
  - Attribute selectors

The visual formatting model of CSS

- Every document element is displayed as a **rectangular box**.
- Each box has a ***content area*** (e.g., text, an image, etc.) and optional surrounding ***padding***, ***border***, and ***margin*** areas:



- Example:

```
/* resulting size of the box: 100px */
#myBox {
    width: 70px;
    margin: 10px;
    padding: 5px;
}
```

Cascade and specificity

Question: **Which rule to apply** if there are more than one **conflicting rules** matching an element?

Every selector has its place in the **specificity hierarchy**. There are **four distinct categories** which define the specificity level of a given selector:

1. Inline styles e.g.: **`<h1 style="color: #fff;">`**
2. ID selectors
3. Classes, attributes and pseudo-classes.
4. Elements and pseudo-elements (# of Element (type) selectors).

Example: The selector

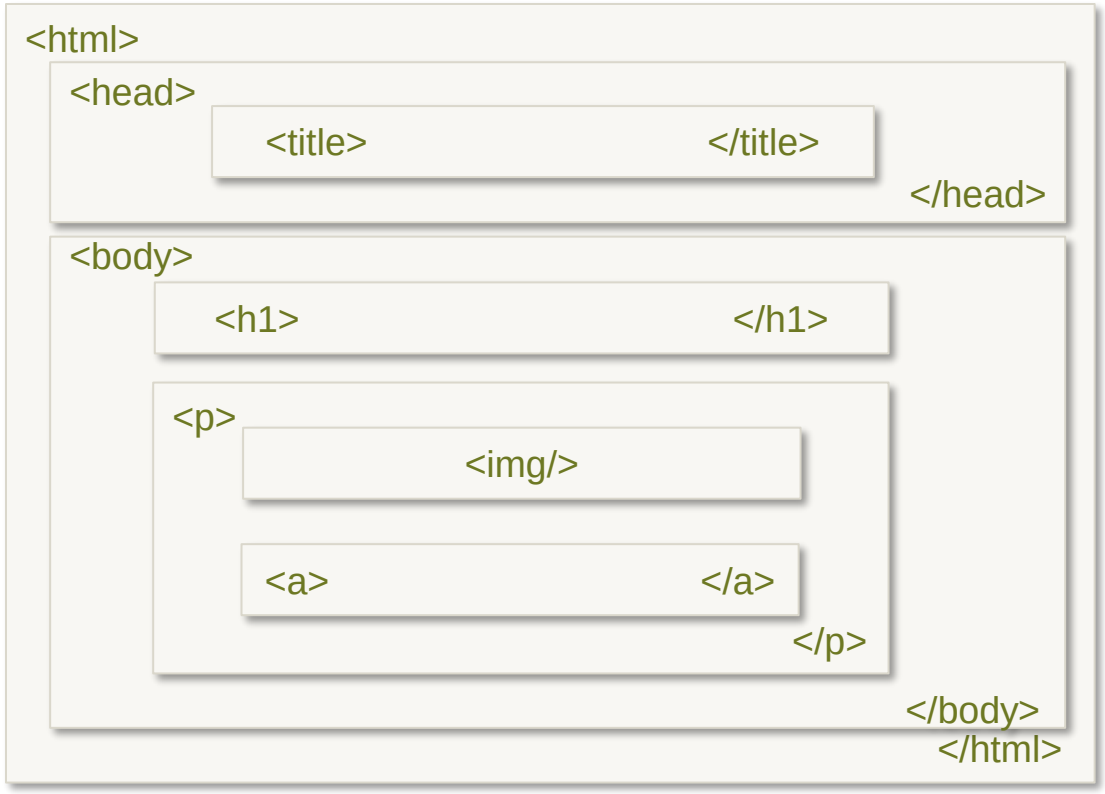
```
body #content .data img:hover { … }
```

has specificity 0,1,2,2 (0100 for #content, 0010 for .data, 0010 for :hover, 0001 for body and 0001 for img)
Equal specificity: **the latest rule is the one that counts**.

(http://www.smashingmagazine.com/2007/07/27/css-specificity-things-you-should-know/)

Inheritance

- Any HTML page comprises a number of elements, which form a containment hierarchy with the <html> element at the top.
- Some properties (font, color, ...) are inherited by the children of an element.



Source: https://www.westciv.com/style_master/academy/css_tutorial/advanced/cascade_inheritance.html

Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

JavaScript Basics

JavaScript is a standard language that allows you to manipulate HTML elements on your web page in any web browser → using JavaScript you are able to obtain a sophisticated cross-platform UI

JavaScript's creator, Brendan Eich, had no choice but to create the language very quickly (or other, worse technologies would have been adopted by Netscape).

Language concepts were borrowed from several programming languages:

- Java (syntax, primitive values versus objects)
- Scheme and AWK (first-class functions)
- Self (prototypal inheritance)
- Perl and Python (strings, arrays, and regular expressions)

→ JavaScript enables a programming style that is a mixture of functional programming (higher-order functions; built-in map, reduce, etc.) and object-oriented programming (objects, inheritance).

History of JavaScript

- *JavaScript* (*LiveScript*) was developed by Brendan Eich at Netscape as the in-page scripting language for Navigator 2 (December 1995)
- Standardization of JavaScript as *ECMAScript* in June 1997
- *ECMAScript* is the official name for JavaScript. A new name became necessary because there is a trademark on *JavaScript* (held originally by Sun, now by Oracle).
- Browser support for different versions is no longer an issue, since introduction of next generation js-compilers, e.g. Babel (syntax transformers, pollyfills)

Version	Year	Features
1	1997	First standard for JavaScript
2	1998	Alignment to ISO/IEC 16262 international standard
3	1999	Added regular expressions, string handling, control statements
4	-	Due to political reasons, there was no agreement on version 4
5	2009	Adds “strict mode”, better support for reflection
6	2015	Added the concepts of classes and modules, collections (e.g., maps and sets), operator overloading, …
7	2016	Added the exponentiation operator and includes operator in arrays
8	2017	Added async/await constructions
9	2018	Added rest/spread operators, asynchronous iteration, and RegEx additions
10	2019	Added Array.prototype.flat, Array.prototype.flatMap, …
11	2020	Introduced BigInt primitive type, …

https://en.wikipedia.org/wiki/ECMAScript

The TC39 Process

- JavaScript is evolved by the TC39 committee
- Problem with ES6 was its scope --> It was released almost 6 years after ES5
- Therefore from now on, releases will happen more frequently (one per year) and follow the new TC39 process

Stage	Purpose	Features
0	strawman	Free-form ideas, reviewed in TC39 meetings
1	proposal	Formally accepted proposal
2	draft	Has description of syntax and semantics
3	candidate	Specification text complete, has at least 2 implementations
4	finished	Ready for standard, passed unit tests

- Babel allows to enable the different stages in the configuration

http://2ality.com/2015/11/tc39-process.html

The Nature of JavaScript

- **It's dynamic**

  Many things can be changed. For example, you can freely add and remove properties (fields) of objects after they have been created. And you can directly create objects, without creating an object factory (e.g., a class) first.
- **It's dynamically typed**

  Variables and object properties can always hold values of any type.
- **It's functional and object-oriented**

  JavaScript supports two programming language paradigms:

  1) functional programming (first-class functions, closures, partial application via bind(), built-in map() and reduce() for arrays, etc.)
  2) object-oriented programming (mutable state, objects, inheritance, etc.).
- **It fails silently**

  JavaScript did not have exception handling until ECMAScript 3. That explains why the language so often fails silently and automatically converts the values of arguments and operands: it initially couldn't throw exceptions.
- **It's deployed as source code**

  JavaScript is always deployed as source code and compiled by JavaScript engines. Source code has the benefits of being a flexible delivery format and of abstracting the differences between the engines. Two techniques are used to keep file sizes small: compression (mainly gzip) and minification (making source code smaller by renaming variables, removing comments, etc.; see slide JavaScript tools).

Speaking JavaScript. Dr. Axel Rauschmayer: http://exploringjs.com

ECMAScript 2015 (ES6)

Goals

Be a better language for writing:

- Complex applications;
- Libraries (possibly including the DOM) shared by those applications;
- Code generators targeting the new edition.

Improve interoperation, adopting de facto standards where possible:

- Classes: are based on how constructor functions are currently used.

```
class Point {
    constructor(x, y)
    {
        this.x = x;
        this.y = y;
    }
    toString() { return `(${this.x}, ${this.y})`; }
}
class ColorPoint extends Point {
    constructor(x, y, color) { super(x, y); this.color = color; }
    toString() { return super.toString() + ' in ' + this.color; }
}
```

Concole output using the classes:

```
> const cp = new ColorPoint(25, 8,
  'green');
> cp.toString(); '(25, 8) in green'
> cp instanceof ColorPoint true
> cp instanceof Point true
```

Under the hood it's still a function:

```
> typeof Point
'function'
```

http://es6-features.org/

ECMAScript 2015 (ES6)

- Modules

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) { return x * x; }
export function diag(x, y) { return sqrt(square(x)+square(y));}
//------ main.js ------
import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

```
//------ MyClass.js ------
export default class { ··· } // no semicolon! /
/------ main2.js ------
import MyClass from 'MyClass';
const inst = new MyClass();
```

- Arrow functions: usefull for callbacks and lambda functions

```
function Person() {
    this.age = 0;
    setInterval(() => {
        this.age++; // this properly refers to the Person object
    }, 1000);
    console.log(age) // very, very old!
}
```

- Named function parameters: `selectEntries({ start: 3, end: 20, step: 2 });`

http://es6-features.org/

ECMAScript 2015 (ES6)

- Template literals

```
var customer = { name: "Foo" }
var card = { amount: 7, product: "Bar", unitprice: 42 } var message = `Hello
        ${customer.name}, want to buy ${card.amount} ${card.product} for
        a total of  ${card.amount * card.unitprice} bucks?`
```

http://es6-features.org/

JavaScript Transpiler

- Transpiler are a great way to use always the latest JavaScript version, without waiting for browser support
- The most popular transpiler right now is Babel
- Babel can convert JSX syntax and strip out type annotations
- It is also built out of plugins, which allows the composition of individual transformation pipelines
- The compiled code is also debuggable due to source map support

https://babeljs.io/

Microsoft TypeScript & Facebook Flow

There is no static typing in ES6

TypeScript enables static typing since ES5

- TypeScript is similar to ES6, but allows optional type annotations.
- TypeScript compiles into ES5

Alternative is Flow from Facebook:

- Developed specifically for ES6
- Based on the flow analysis

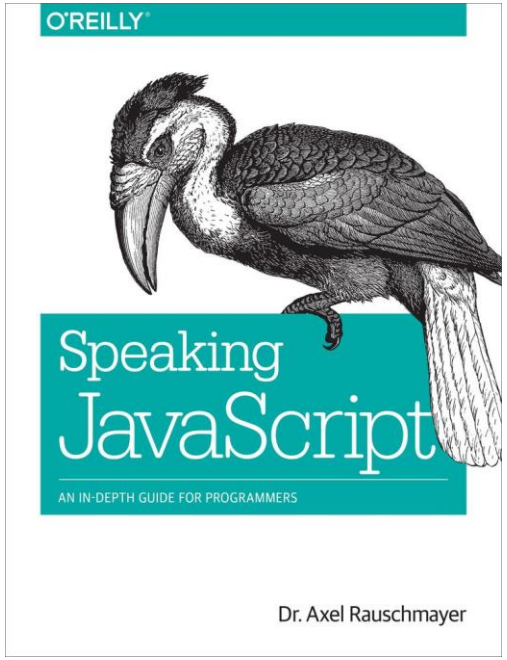
Two benefits of static typing are:

- It allows you to detect a certain category of errors earlier, because the code is analyzed statically (during development, without running code). → It is complementary to testing and catches different errors.
- It helps IDEs with auto-completion.

Exploring ES6. Dr. Axel Rauschmayer: http://exploringjs.com

Recommended literature

- JavaScript books by Dr. Axel Rauschmayer - http://exploringjs.com/
- Books are available for free

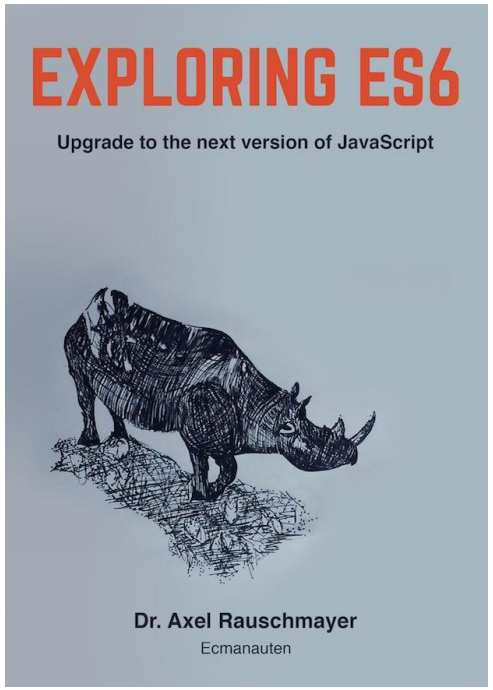


Speaking JavaScript

An in-depth guide to JavaScript.

Covers: All of JavaScript, up to and including ES5

Required knowledge: any programming language (ideally, an object-oriented one)



Exploring ES6

The most comprehensive book on ES6. But don't be intimidated: you decide how deep to go.

Covers: ES6 (ES2015)
Required knowledge: ES5

DOM – Document Object Model

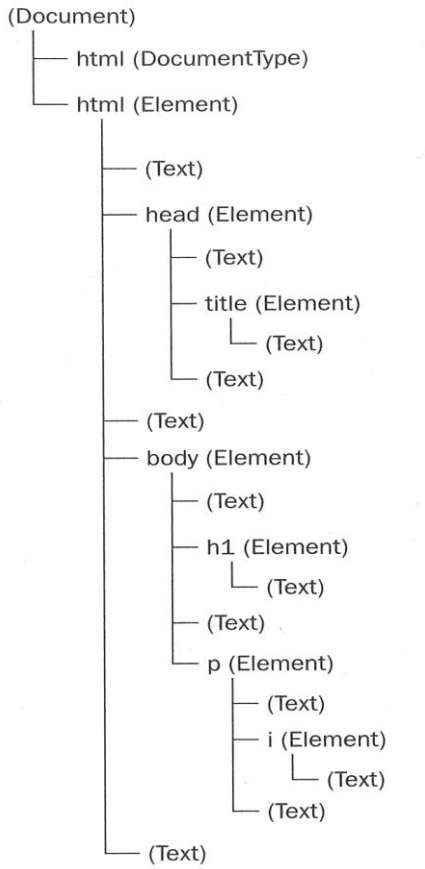
- The Document Object Model is the way JavaScript sees its containing HTML page and browser state.
- The DOM is an **object-oriented representation** of an HTML or XML document.
- The structure of an HTML and XML document is **hierarchical** → the DOM structure is a tree.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
    <title>DOM Example </title>
</head>
<body>
    <h1>Page title </h1>
    <p> Some <i>very</i> unimportant text. </p>
</body>
</html>
```

Page title

Some *very* unimportant text.



DOM API

- The DOM has three levels of support (Level 1, 2, and 3).
- The entry point to the DOM is the **document** object, available to the JavaScript code as a global variable. From there on it is possible to navigate the DOM recursively using attributes like **`firstChild`**, **`lastChild`**, **`childNodes`**, ...

```
html = document.childNotes[1]; //the html element
head = html.firstChild.nextSibling; // the head element
document.parentNode; //Returns null
head.parentNode; //Return the html element
```

- Navigating the DOM this way can be pretty lengthy. There are two methods to access HTML elements by their ID and their tag: **`getElementById`**() and **`getElementByTagName`**()

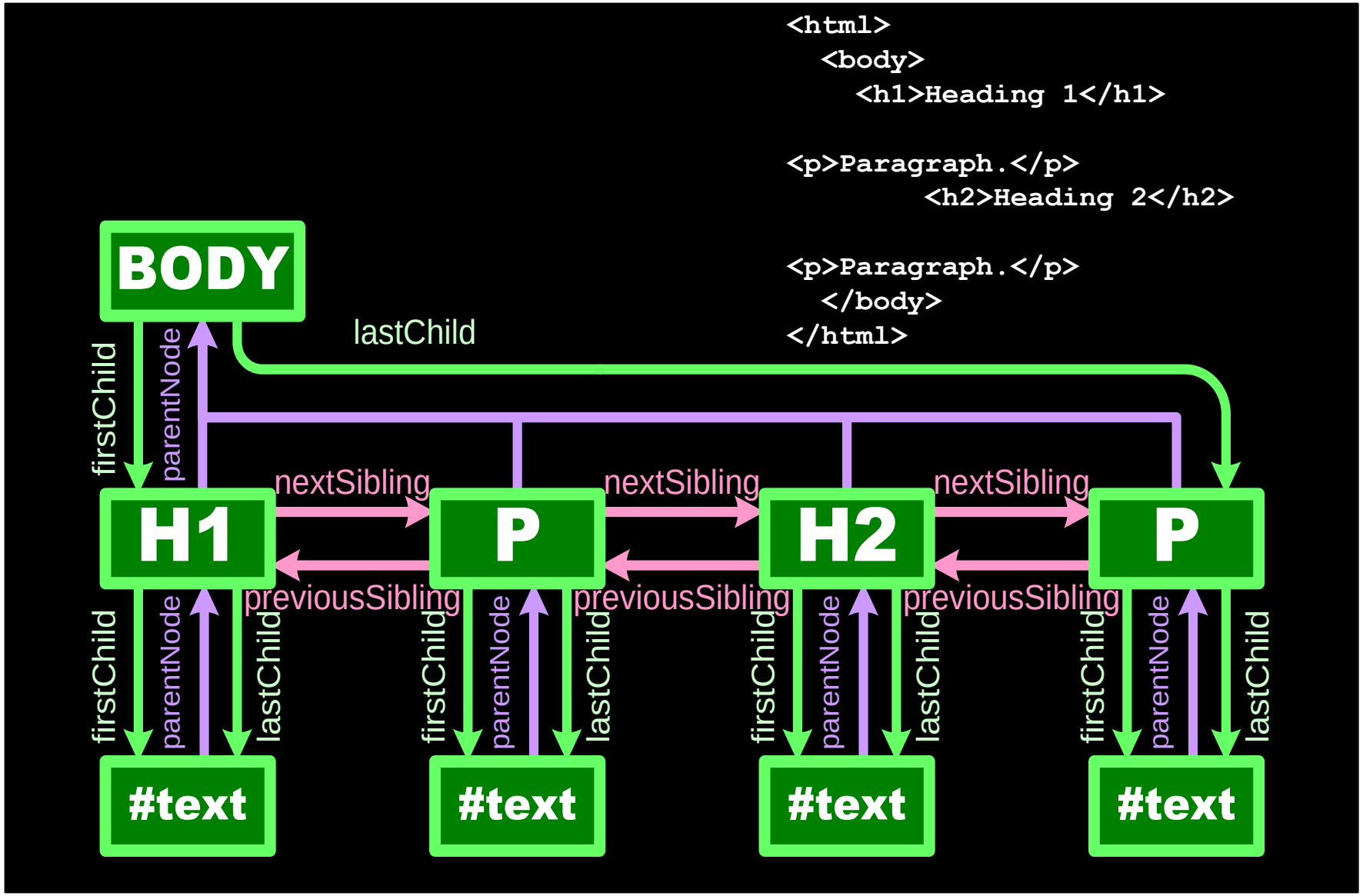
```
unimportantText = document.getElementsByTagName("p")[0];
pleaseNote = document.createElement("p");
pleaseNote.appendChild(document.createtextNode("Please note: "));
unimportantText.parentNode.insertBefore(pleaseNote, unimportantText);
```

Page title

Please note:

Some *very* unimportant text.

Main Navigation Axis of the DOM



[http://yuiblog.com/assets/crockford/theory.zip]

Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Asynchronous operations

- Operations occurring independent of the main program flow

- Example problem:
  - Weather application
  - HTTP request takes time
  - Application needs to wait, obviously it can continue with other tasks such as rendering
  - Upon response, application needs to handle the response data

- Other examples:
  - Database query
  - Client requests user data from server
  - Reading/writing files
  - …
  - Asynchronous operations are required in almost any application

- JavaScript is non-blocking

- JavaScript offers three options:
  - Callbacks
  - Promises
  - async/await

The Concept of Callbacks in JavaScript

- A function which is executed once the result of an asynchronous operation is ready
- Therefore, the asynchronous operation takes (at least) one function as an argument, which is the callback
- Job to be executed by the callback function is put on a queue and executed in an “event-loop”

```
const request = require('request');

let result;

function handleResponse(error, response, body) {
    if(error){
        // Handle error.
    }
    else {
        result = body
    }
}

request('https://www.weather.com', handleResponse);

console.log(result); // Prints "undefined", since the Request has not been completed once this line is called
```

Common node.js module for http requests

The Callback Hell

- Chaining: What if you need to perform another asynchronous operation with the result of the first one?
- The result of the asynchronous operation is only available in the callback
- Code becomes less readable

```
const request = require('request');

request('http://www.weather.com', function (firstErr, firstResp, firstBody) {
    if(firstErr) {
        // Handle error.
    }
    else {
        request(`http://www.weather.com/${firstBody.value}`, function (secondErr, secondResp, secondBody) {
            if(secondErr) {
                // Handle error.
            }
            else {
                // Use secondBody for something
            }
        });
    }
});
```

Result from the first call

The Concept of Promises in JavaScript

- **Promises in JavaScript** allow to encapsulate asynchronous operations and provide a way:
  - to handle errors in natural way
  - to write cleaner code:
    - without having callback parameters
    - without modifying the underlying architecture

- Instead of providing a callback, a Promise has it's own methods which can be called to define what to do upon completion
  - **then(…)** is used for when a successful result is available
  - **catch(…)** is used for when something went wrong

```
const axios = require('axios');

axios.get('http://www.weather.com')
    .then(function(response) {
        // Handle the response
    }).catch(function (error) {
        // Handle error
    });
```

Common node.js module for http requests, using Promises

This is not a promise, but a function returning a Promise

Chaining of Promises in JavaScript

- You can chain several asynchronous operations

- *then(…)* must return either another promise or a value/object
  - In case of a promise, the following *then(…)* is called once the promise is resolved
  - In case of an object, the following *then(…)* receives the object as an argument

- One *catch(…)* at the end of the chain is sufficient

```
const axios = require('axios');

axios.get('http://www.weather.com')
    .then(function(response) {// Response being the result of the first request
        // Returns another promise to the next .then(..) in the chain
        return axios.get(`http://www.somepage.com/${response.someValue}`);
    }).then(function(response) { // Response being the result of the second request
        // Handle response
    }).catch(function (error) {
        // Handle error.
    });
```

Values are still only accessible within the chain

Wrapping Callback-based APIs with Promises

- Callbacks and Promises are not interchangeable

- A Promise constructor takes a function as an argument and that function gets passed two callbacks:
  - one for notifying when the operation is successful (*resolve*)
  - and one for notifying when the operation has failed (*reject*).
  - The argument passed when calling resolve will be passed to the next *then()* in the promise chain
  - The argument passed when calling reject will end up in the next *catch()*

```
function getWeather() {
    return new Promise(function(resolve, reject) {
        request('https://www.weather.com', function(error, response, body) {
            if(error) {
                reject(error);
            }
            else {
                resolve(body);
            }
        })
    });
}
```

```
// Calling resolve in the Promise will get us here, to the then(…)
getWeather('someValue').then(function(result) {
    // Handle result
 console.log('success');
})
// Calling reject in the Promise will get us here, to the catch(…)
.catch(function(error){
    // Handle error
 console.log('error');
});
```

The Concept of async/await in JavaScript

- async/await is a language feature that is a part of the ES8 standard
- async/await is the next step in the evolution of handling asynchronous operations in JavaScript
- It provides two new keywords to use: *async* and *await*
  - *async* is for declaring that a function will handle asynchronous operations
  - *await* is used to declare that we want to “await” the result of an asynchronous operation inside a function that has the async keyword
- async/await produces more readable code, looking like synchronous/procedural code

Using await

- A function call is only allowed to have the *await* keyword, if the function being called is “awaitable”
- A function is “awaitable” either if it has the *async* keyword or if it returns a Promise

getWeather() from the previous slide (Slide 42)

```
async function demo1() {
    let result = await getWeather('https://www.weather.com');
    console.log(result);
}

async function demo2() {
    try {
        let result = await getWeather('https://www.weather.com');
        console.log(result);
    }
    catch(error) {
        console.log(error);
    }
}
```

Returns the value, which would be in the *then(…)* block of the Promise returned by *getWeather(…)*

The value, which would be in the *catch(…)* block of the Promise returned by *getWeather(…)*

```
function fetchTheCityID(cityCode) {
    return axios.get(`https://www.weather.com/${cityCode}`);
}
function fetchWeather(id) {
    return axios.get(`https://www.weather.com/weather/${id}`);
}
async function demo3(cityCode) {
    const id = await fetchTheCityID(cityCode);
    const result = await fetchWeather(id);
    console.log(result);
    return result;
}

async function demo4(cityCode) {
    try {
        const id = await fetchTheCityID(cityCode);
        const result = await fetchWeather(id);
        console.log(result);
        return result;
    }
    catch(error) {
        console.log(error);
    }
}
```

Error handling is done with try/catch, while one catch block is sufficient (as done with Promises)

Using async

- Functions with the *async* keyword are interchangeable with functions returning a Promise
- An *async* function returns always a promise (values will be transformed)

Proper way to utilize *await*

```
async function getMyWeather() {
    try {
        let result = await getWeather("https://www.weather.com");
        console.log(result);

        return result;
    }
    catch(error) {
        console.log(error);
    }
}
```

throw error;

Promise is waived

```
getMyWeather();

getMyWeather().then(function(result) {
    console.log(result);
});

getMyWeather().then(function(result) {
    console.log(result);
}).catch(function(error) {
    console.log(error);
});
```

Errors are ignored, while *getMyWeather()* will always resolve a Promise into *.then()*

Makes only sense in case…

Outline

4. Developing Serverless Single-Page Web Applications

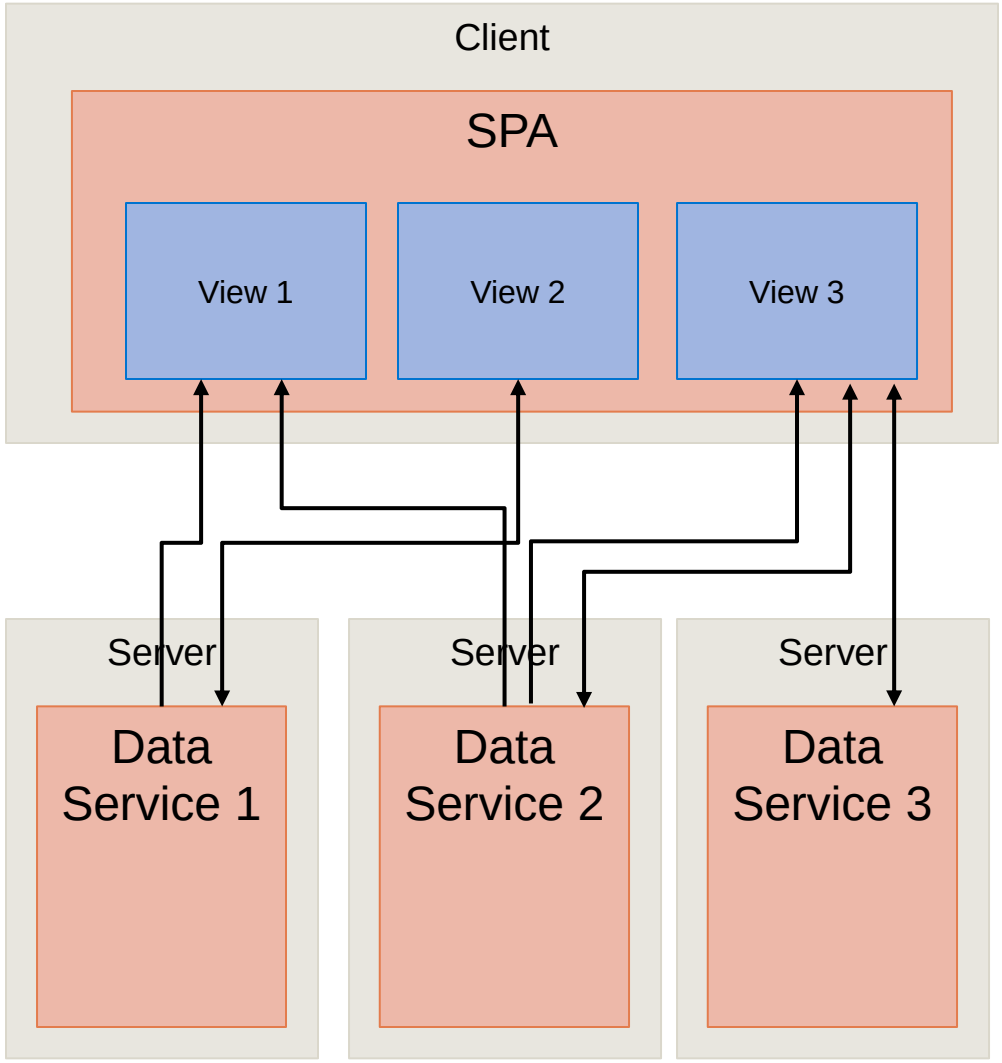
- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Single-Page Applications (SPA)

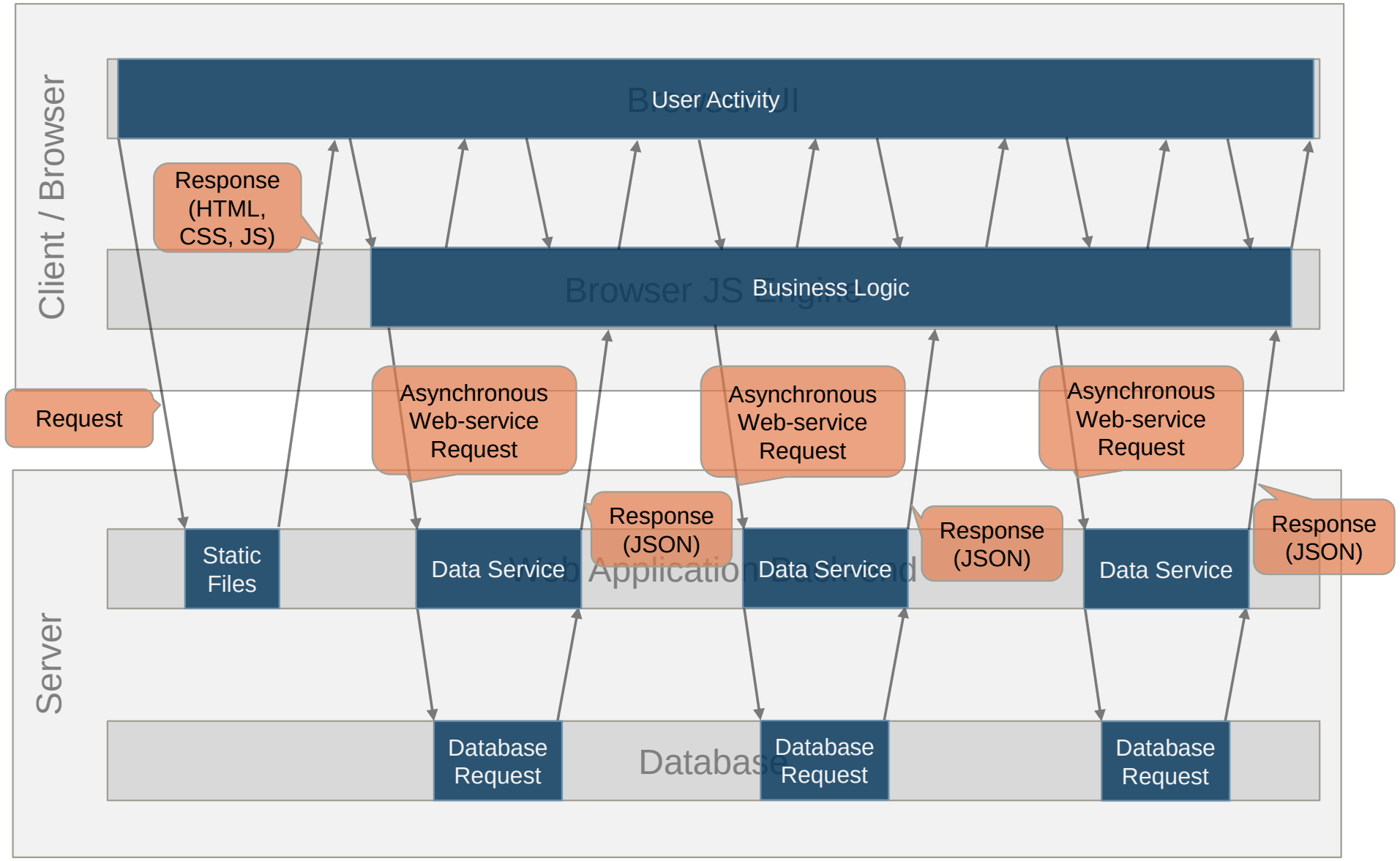
- With one request, the web server serves a "single page" which either
  - already contains all views including HTML templates, CSS style sheets, and JS logic, or
  - dynamically loads required resources

- The **actual data** is loaded **asynchronously**, typically via a **REST API**
- For users, SPAs feel much more like **desktop or mobile applications** rather than websites

- Different SPA architecture styles
  - **"Thick" stateful server architecture**: Most of the business logic is implemented at the server. There is also a server-side session state.
  - **"Thick" stateless server architecture**: Most of the business logic is implemented at the server. All requests are stateless.
  - **"Thin" server architecture or "serverless" single page application**: Most of the business logic moves to the client, i.e., web server turns into a data service. All requests are stateless.



"Serverless" Single-Page Application Architecture



Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

[Web] component-based architecture as a trend in client-side development

A **component** in software engineering is an element that bundles a **set of related functions and data**. Each component is a self-contained piece of software. A component-based architecture emphasizes the **separation of concerns** and the **single responsibility** principle.

Reasons to be popular in client-side development:

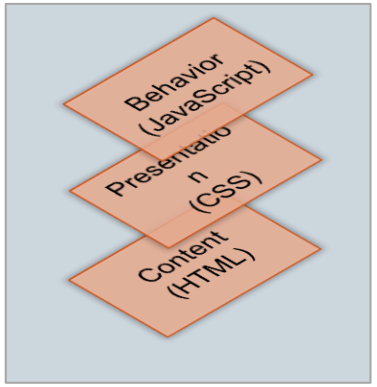
- Business logic became part of the client-side implementation
- Increased complexity of UI

Both resulted in

- Increase in number of lines of code per one JavaScript file
- Data flow is hard to understand and control
- Parts of functionality impossible to reuse
- Functionality is difficult to test

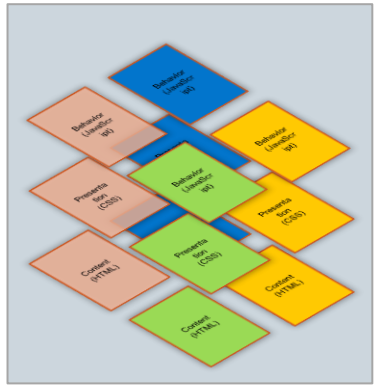
Solution: Divide the whole application into the set of independent components, each with a single responsibility.

Content layers



How to divide them correctly?

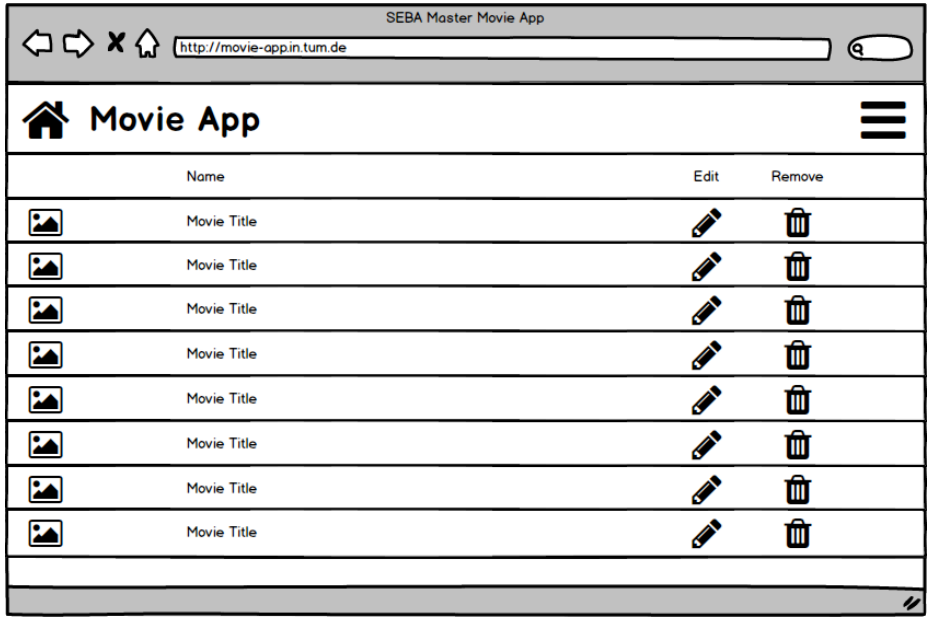
Components (for each component its own content layers defined)



Thinking in web components: from mockup to web components

MovieApp example:

- Single page application
- Unauthorized user is able to see the list of movies
- Unauthorized user is able to see detailed information about movie
- Authorized user is able to create, edit and delete new movie

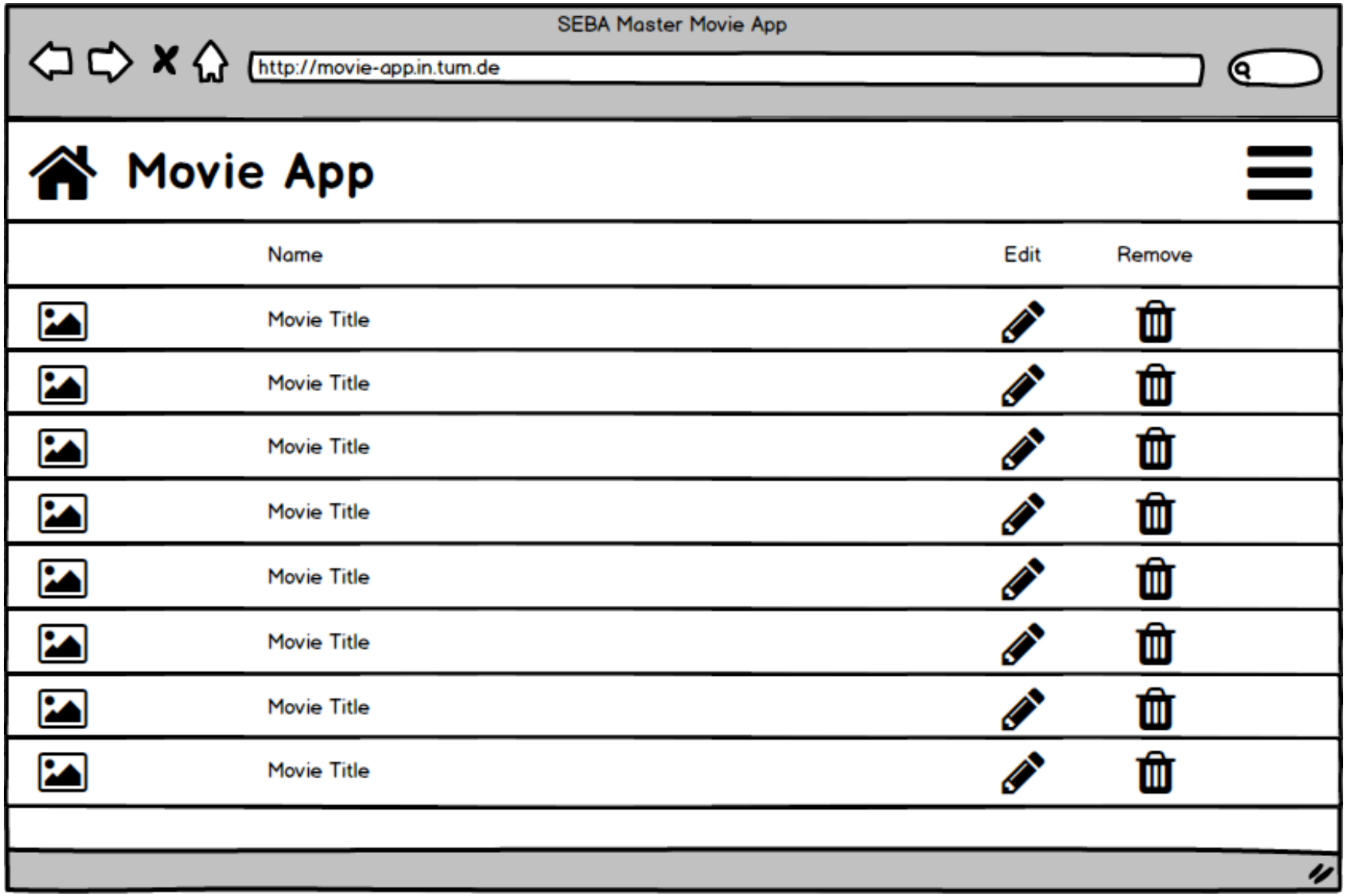


From Mockup to Web Components:

1. Break the UI into a component hierarchy
2. Build a static version of your components
3. Identify the minimal (but complete) representation of UI state
4. Identify the data flow between components

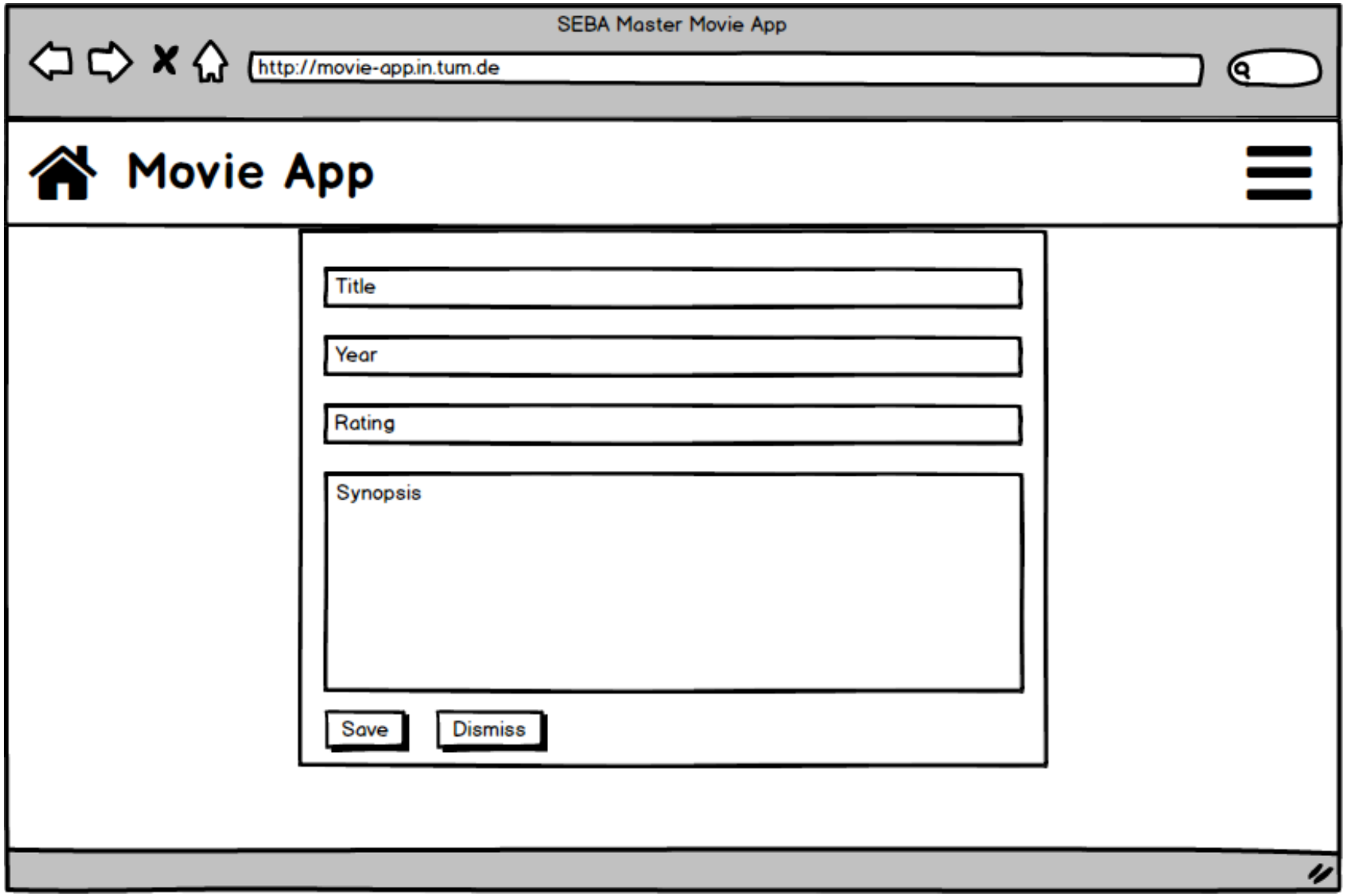
From Mockups to Web Components

List of movies

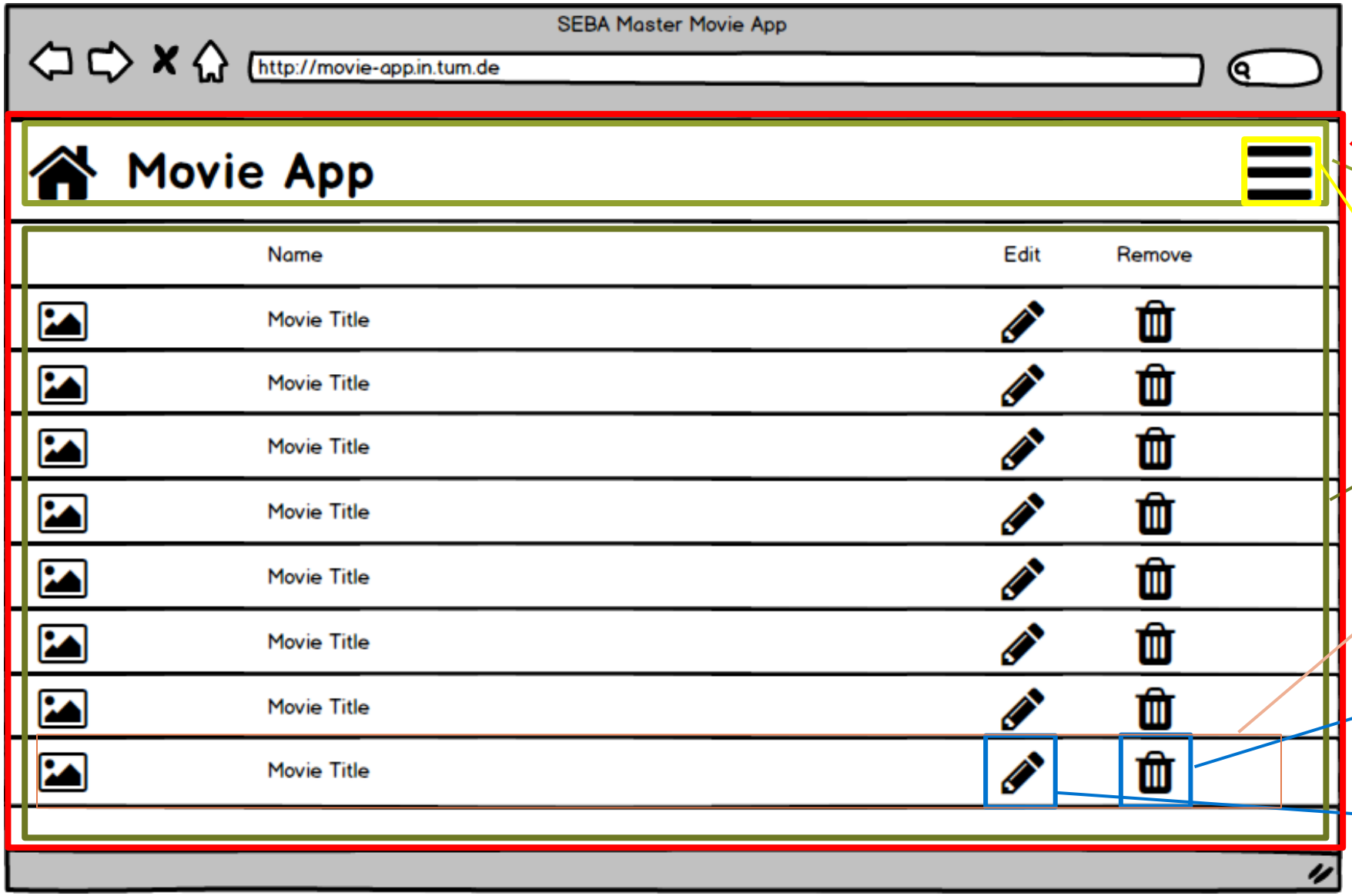


From Mockups to Web Components

Adding new video



1. Break the UI into a component hierarchy



Page

Header

KebabMenu*

MovieList

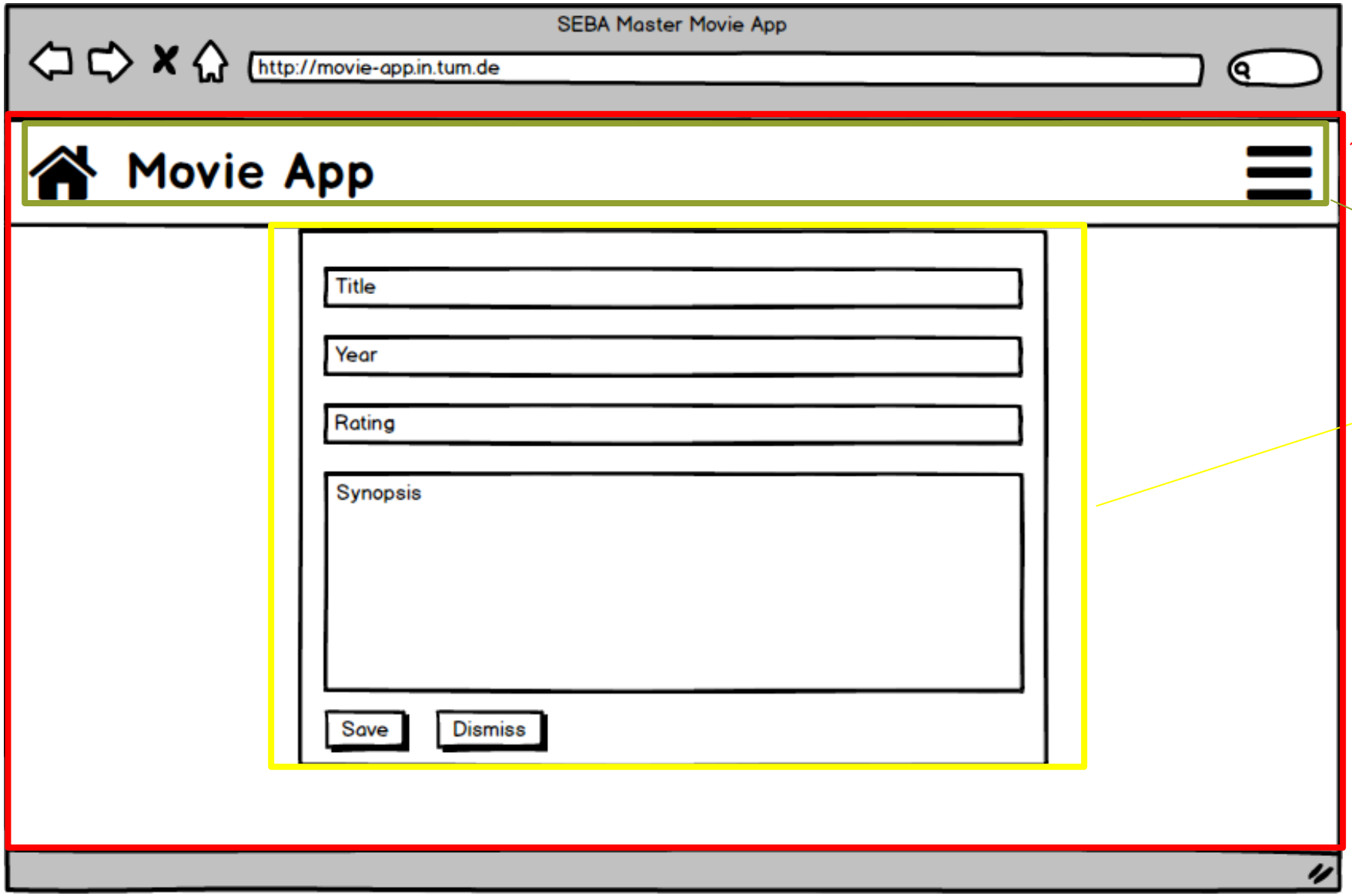
MovieListRow

DeleteMovieButton

EditMovieButton

*Often referred to as Burger Menu

1. Break the UI into a component hierarchy



Page

Header

MovieForm

1. Break the UI into a component hierarchy

Based on the mockup the following hierarchy is produced:

- Page
  - Header
    - Kebab Menu
    - Existing react-md components
  - MovieList
    - MovieListRow
      - EditMovieButton
      - DeleteMovieButton
  - MovieForm

2. Build a static version of your components

Each of the components will contain three parts:

1) HTML file
2) CSS file
3) JS file

In this step, the HTML template and CSS styles for each of the component has to be created

- Page
  - Header
    - Kebab Menu
    - Existing react-md components
  - MovieList
    - MovieListRow
      - EditMovieButton
      - DeleteMovieButton
  - MovieForm

3. Identify the minimal (but complete) representation of UI state

At this point, the behavior of the components has to be described and later expressed as JS methods

- *Page*
  - *Header*
    - *KebabMenu:* Show add movie and logout buttons if user is logged in, otherwise show login button
  - *MovieList*: show the list of all movies
    - *MovieListRow:* show the information concerning a movie
    - *EditMovieButton*: redirect user to *ViewMovieEditComponent* if user is logged in, otherwise redirect user to the *LoginComponent*
    - *DeleteMovieButton*: delete movie if user is logged in, otherwise redirect user to the *LoginComponent*
  - *MovieForm*: render *MovieForm*

4. Identify the data flow between components

Step 1-3 are well-understood and followed in very similar ways in various client-side frameworks.

There are different paradigms followed by client-side frameworks regarding when and how the changes in one component can cause changes in other components

→ two-way data binding (Angular), one-way data binding (ReactJS).

In chapter 5, we will explain in detail the ReactJS approach to this problem.

Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Component-based frameworks

The concept of web components is central to recent web frameworks for client-side development:

- ReactJS
- Angular
- Vue.js
- Cycle.js
- Lit

What are the differences among them?

- **Implementations differ**
  - Angular (routers, directives, templates, services, middleware)
  - React (states, events, virtual DOM, middleware)
  - …
- **Level of granularity ("convention over configuration"):**
  - **Angular**
    → Provides you with out-of-the-box setup for binding your data, business logic and view.
  - **React**
    → Only focuses on view layer.
    → Data bindings and logic has to be configured separately using Redux or Flux or …
- **Community support (seed projects, documentation, tutorials)**

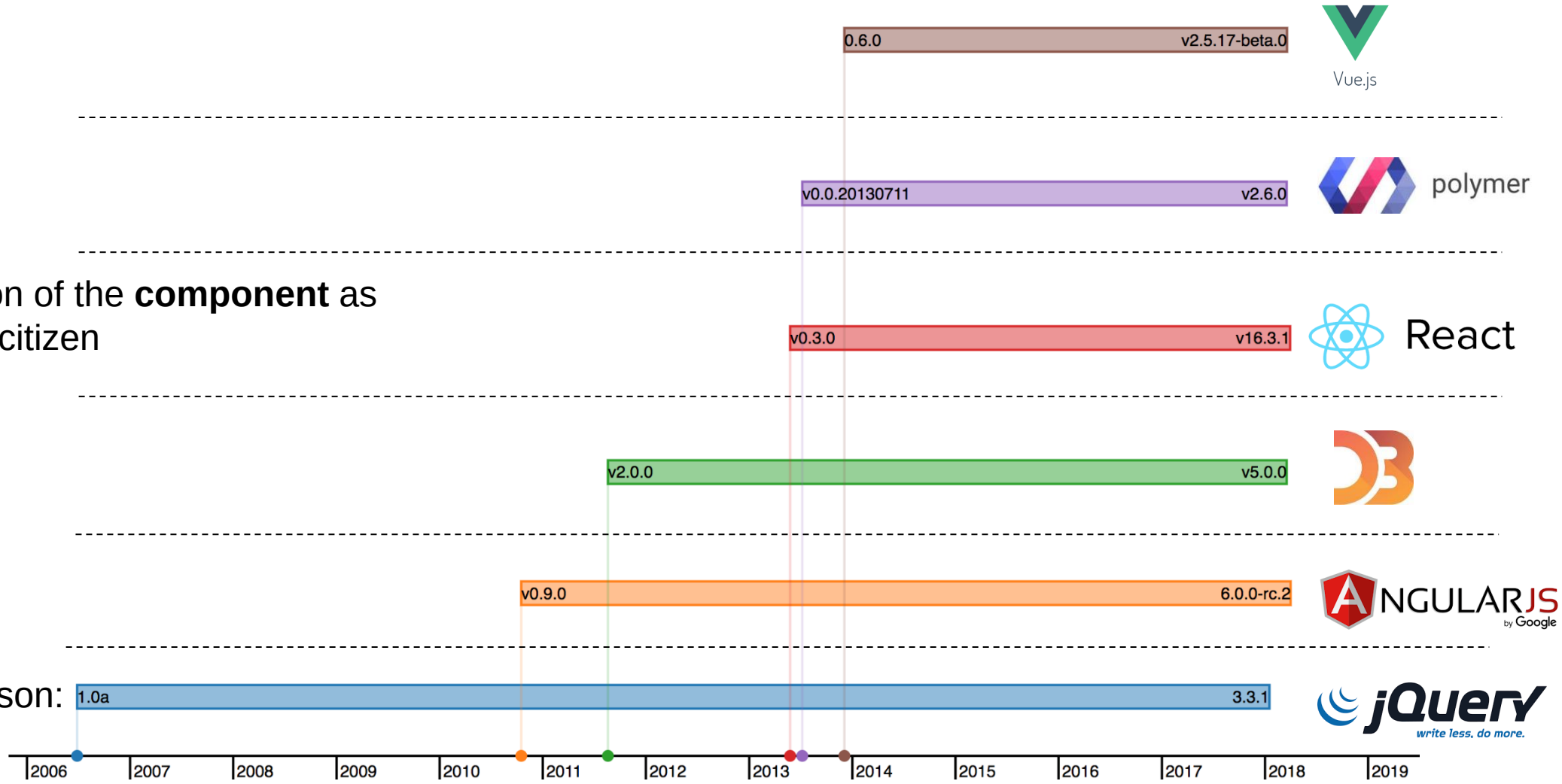
Since 2014 the W3C has been working on a draft specification for web components to be supported natively by future browsers.

Evolution of component-based frameworks

Overview

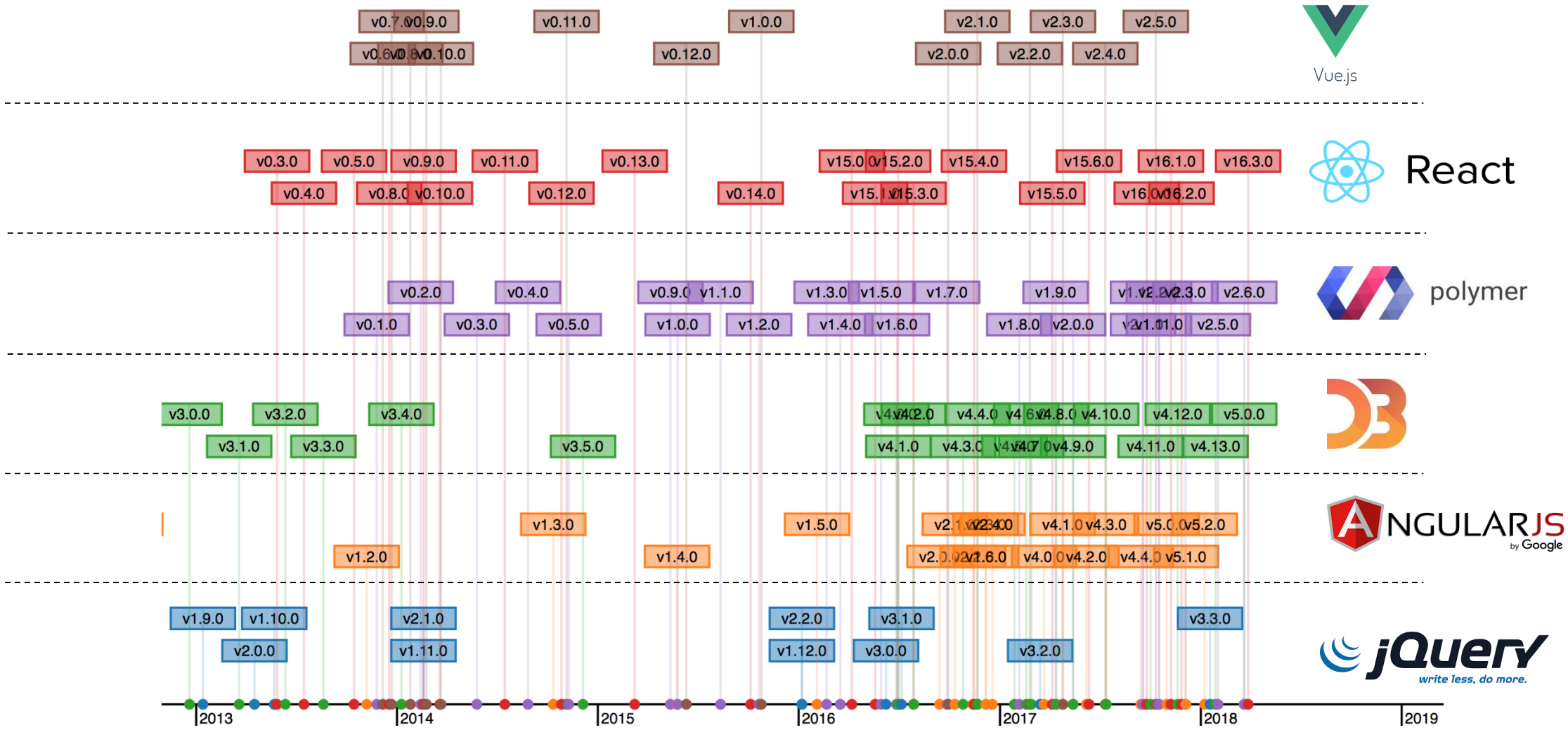
Introduction of the **component** as first-class citizen

For comparison:



Component-based frameworks

Version evolution



Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Simplifying content manipulation

Using the DOM API directly is **cumbersome and error prone** (see also "An Inconvenient API: The Theory of the DOM")

- Different browsers provide different DOM APIs
- The methods provided by the DOM API for navigating and manipulating the DOM are not very comfortable

A JavaScript platform library **insulates the application from the poisonous browsers**.

Examples: **JQuery**, D3.js, Dojo, Prototype, Mochikit, YUI

Overview of jQuery

jQuery: The Write Less, Do More, JavaScript Library [http://jquery.com/]

- Browser compatibility: Firefox 1.5+, Internet Explorer 6+, Safari 2.0.2+, Opera 9+
- jQuery is available under both MIT and GPL licenses.
- The development of jQuery started in August 2005

Google Trend for “jQuery” as of May 2021

Basics of jQuery

Load the jQuery library

```
<html>
<head>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      // we will add our javascript code here
    </script>
</head>
<body>
    <!-- we will add our html content here -->
</body>
</html>
```

Register an event for all <a> tags in a document

```
jQuery(document).ready(function () {
    jQuery("a").click(function () {
        alert("Hello world!");
    });
});
```

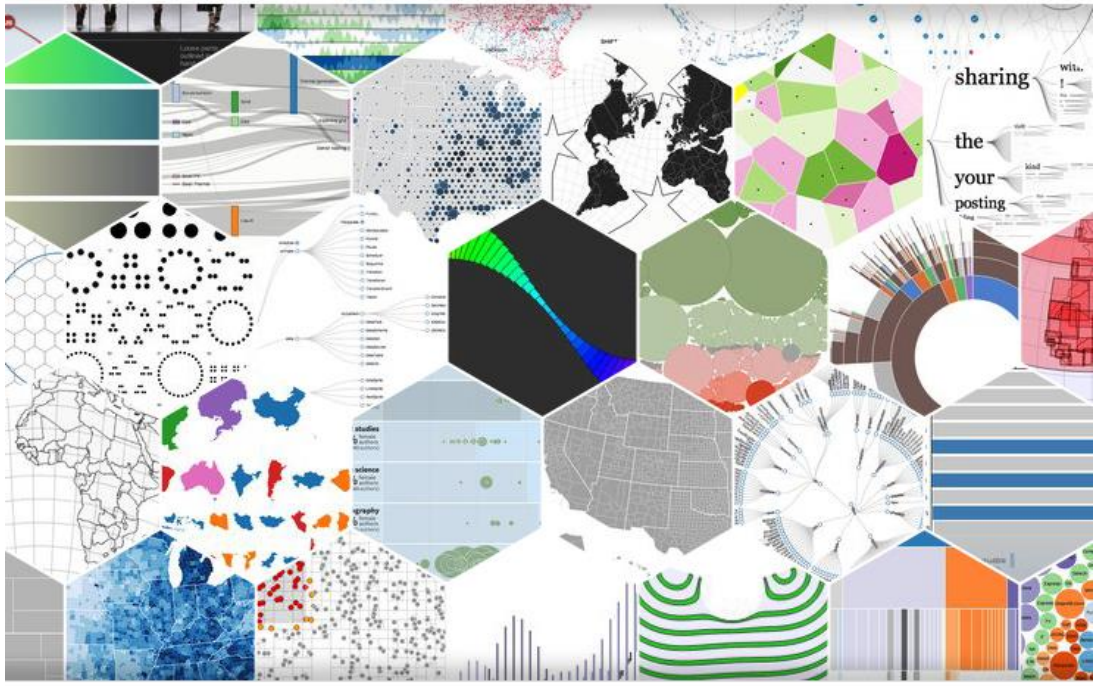
Using selectors to find elements

```
$(document).ready(function () {
    $("#orderedlist > li").addClass("blue");
});
```

```
$(document).ready(function () {
    $("a").hover(function () {
        $(this).parents("p").addClass("highlight");
    }, function () {
        $(this).parents("p").removeClass("highlight");
    });
});
```

[http://learn.jquery.com/about-jquery/how-jquery-works/]

Data-Driven documents (D3.js)



- D3.js is a JavaScript library for manipulating documents based on data.
- D3 helps you bring data to life using HTML, SVG, and CSS.
- D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

Some key features of D3.js

Dynamic properties

- Styles, attributes, and other properties can be specified as functions of data, not just simple constants.

```
d3.selectAll("p").style("color", function () {
    return "p" + Math.random() * 360 + "p";
});
```

Enter and exit

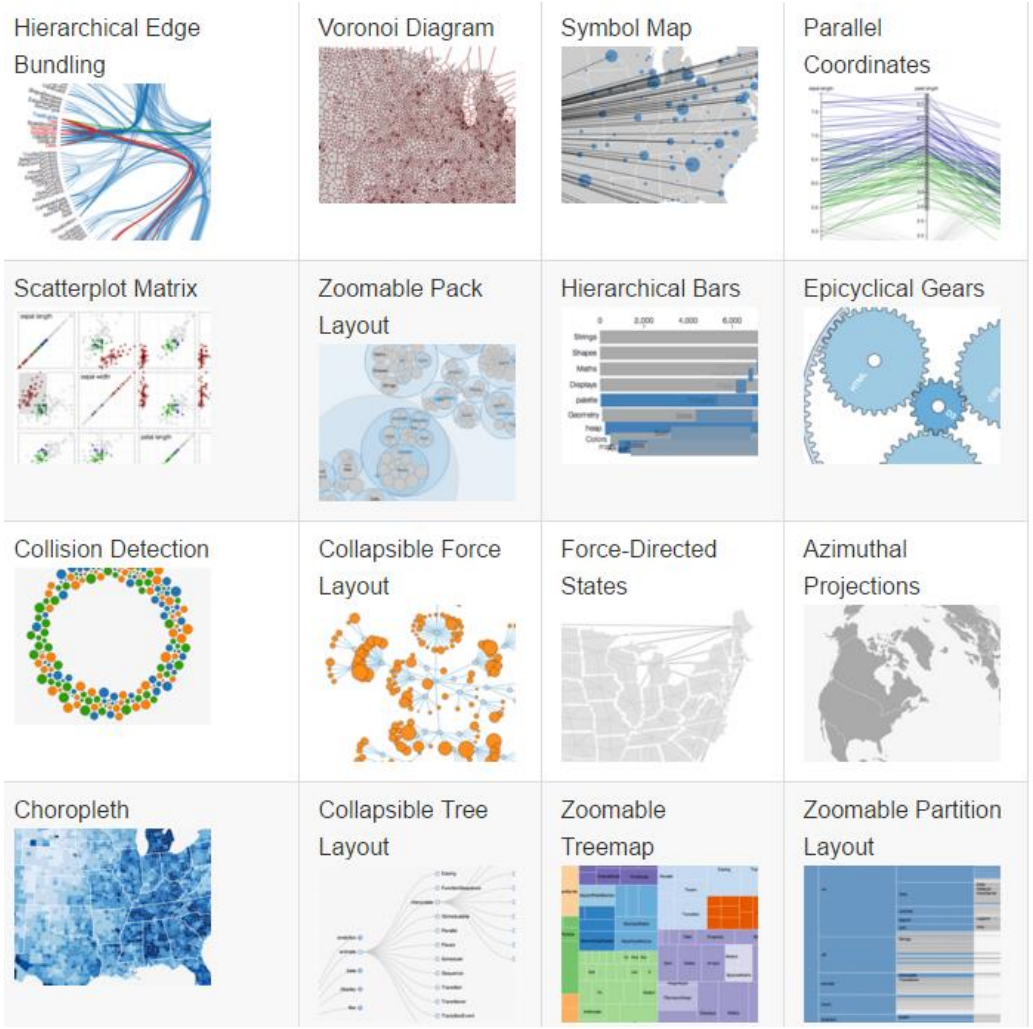
- Allows to create new nodes for incoming data and remove outgoing nodes that are no longer needed.
- When data is bound to a selection, each element in the data array is paired with the corresponding node in the selection. If there are fewer nodes than data, the extra data elements form the enter selection.

```
d3.selectAll("p")
    .data([4, 8, 15, 16, 23, 42])
    .enter()
    .append("p")
    .text(function(d) {
        return "I'm number " + d + "!";
    });
```

Transitions

- Transitions gradually interpolate styles and attributes over time.

D3.js – Reusable examples



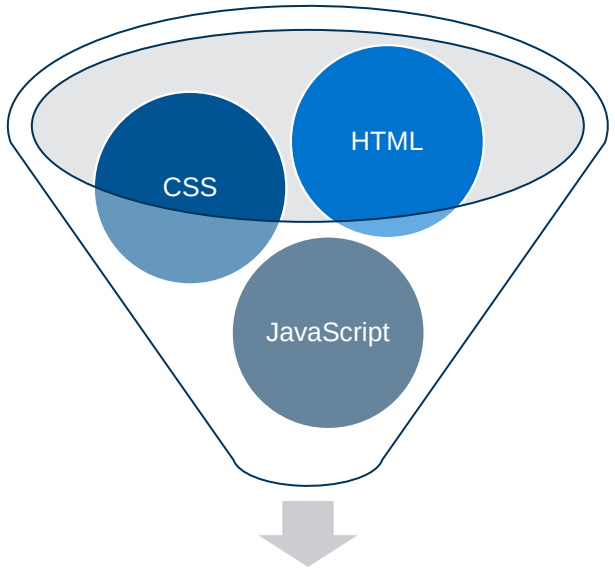
[https://github.com/mbostock/d3/wiki/Gallery]

Outline

4. Developing Serverless Single-Page Web Applications

- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

UI design frameworks



Reusable UI components

- UI design frameworks for the web contain reusable UI components
- Generic solution for the repetitive tasks in the process of design web applications such as:
  - Normalize the style sheet
  - Layouts
  - Responsive designs
  - Web typography and icons
  - Extensions for the basic input components (e.g. tooltips, buttons, date, etc)

Bootstrap: Design templates for web applications

- Free collection of tools for web site creation
- Initial release in August 2011 by Twitter

Motivation for Twitter Bootstrap

- Design templates for typography, forms, buttons, navigation as well as extensions written in the JavaScript language
- Provide a single library for interface development to avoid inconsistencies in the development
- Support for responsive design of web sites

```
<script src=“scripts/jquery.js" type="text/javascript"></script>
<script src="scripts/bootstrap.js" type="text/javascript"></script>
<script src="scripts/popper.js" type="text/javascript"></script>
<link href=“styles/bootstrap.css" rel="stylesheet" type="text/css" />
```

http://getbootstrap.com/getting-started/

Bootstrap – Layouts

Grid system

- The Bootstrap grid system uses a series of containers, rows, and columns and is built with “flexbox”
- Bootstrap is fully responsive and works on all devices
- There are various column classes
  - >= 768px: *col-md-**
  - < 576px: *col-xs-**

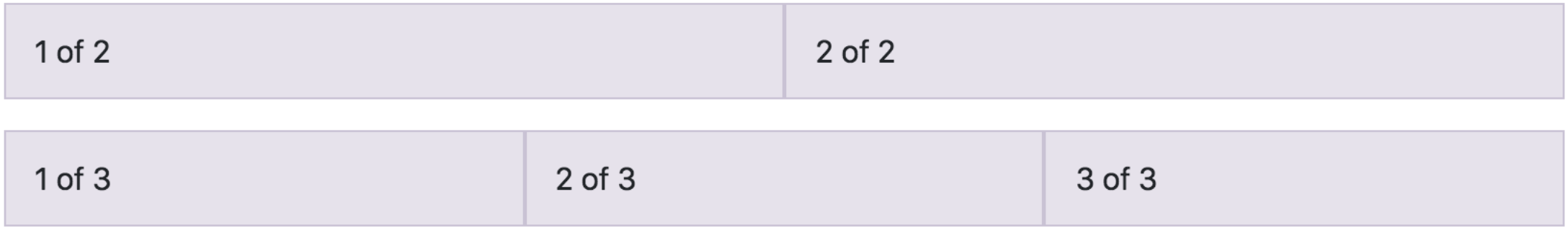
http://getbootstrap.com/getting-started/

Bootstrap – Layouts

Grid system

- You can also rely on the auto-layout features of Bootstrap

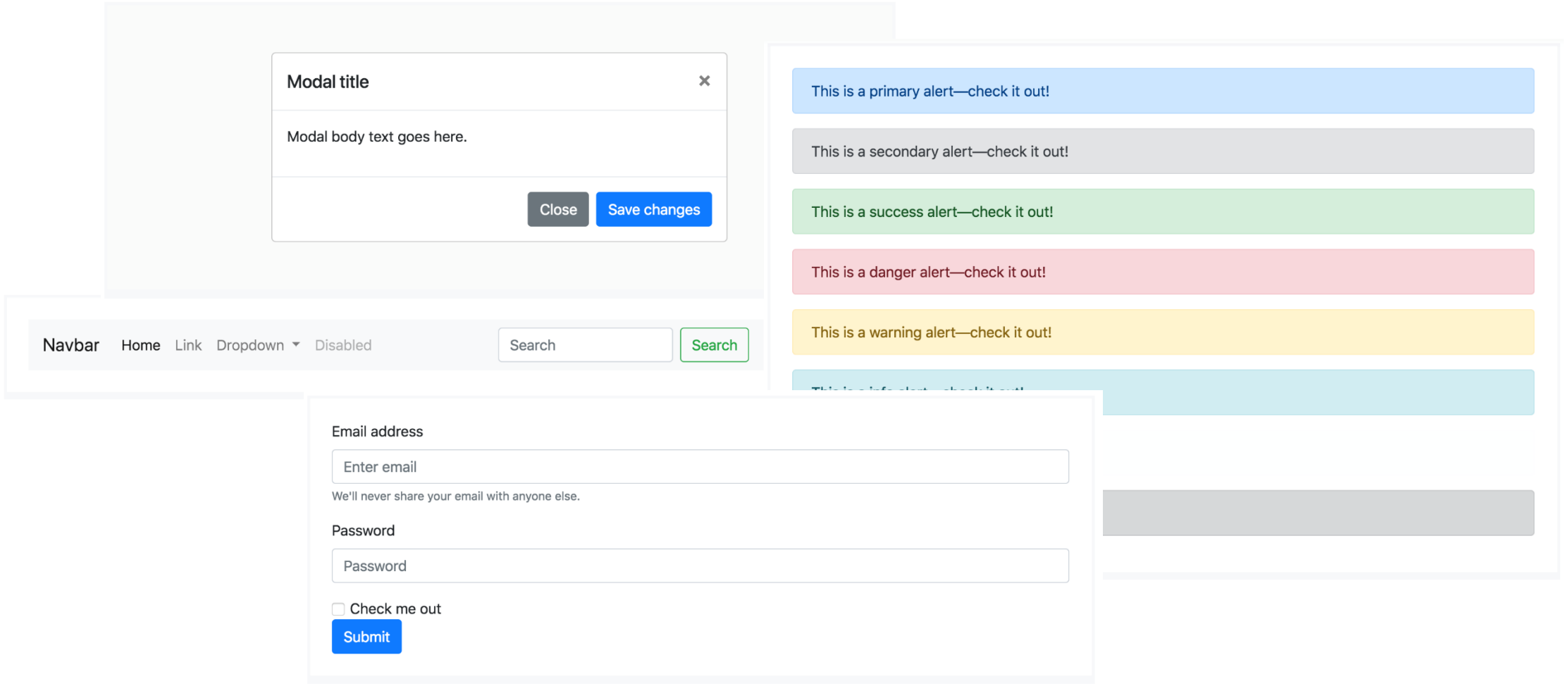
```
<div class="container">
        <div class="row">
                <div class="col"> 1 of 2 </div>
                <div class="col"> 2 of 2 </div>
        </div>
        <div class="row">
                <div class="col"> 1 of 3 </div>
                <div class="col"> 2 of 3 </div>
                <div class="col"> 3 of 3 </div>
        </div>
</div>
```



http://getbootstrap.com/getting-started/

Bootstrap – Basic components

Bootstrap provides basic UI components and themes, e.g., navbars, carousels, blogs, dashboards, etc.



Bootstrap - Example

Home

Home / Library

Home / Library / Data

```
<nav aria-label="breadcrumb">
        <ol class="breadcrumb">
                <li class="breadcrumb-item active" aria-current="page">Home</li>
        </ol>
</nav>
<nav aria-label="breadcrumb">
        <ol class="breadcrumb">
                <li class="breadcrumb-item"><a href="#">Home</a></li>
                <li class="breadcrumb-item active" aria-current="page">Library</li>
        </ol>
</nav>
<nav aria-label="breadcrumb">
        <ol class="breadcrumb">
                <li class="breadcrumb-item"><a href="#">Home</a></li>
                <li class="breadcrumb-item"><a href="#">Library</a></li>
                <li class="breadcrumb-item active" aria-current="page">Data</li>
        </ol>
</nav>
```

CSS classes as provided by Bootstrap

Material Design

- Material Design is a design language developed in 2014 by Google
- Initially announced on June 25, 2014 at the Google I/O conference
- Material-UI consists of React components implementing Google's Material Design

Motivation for Material-UI Design

- Material Design is a design language for the users of the Google Ecosystem that synthesizes the classic principles of good design with the innovation and possibility of technology and science.
- Material-UI implements Material Design components as React components
  - In contrast, Bootstrap is based on CSS and HTML templates

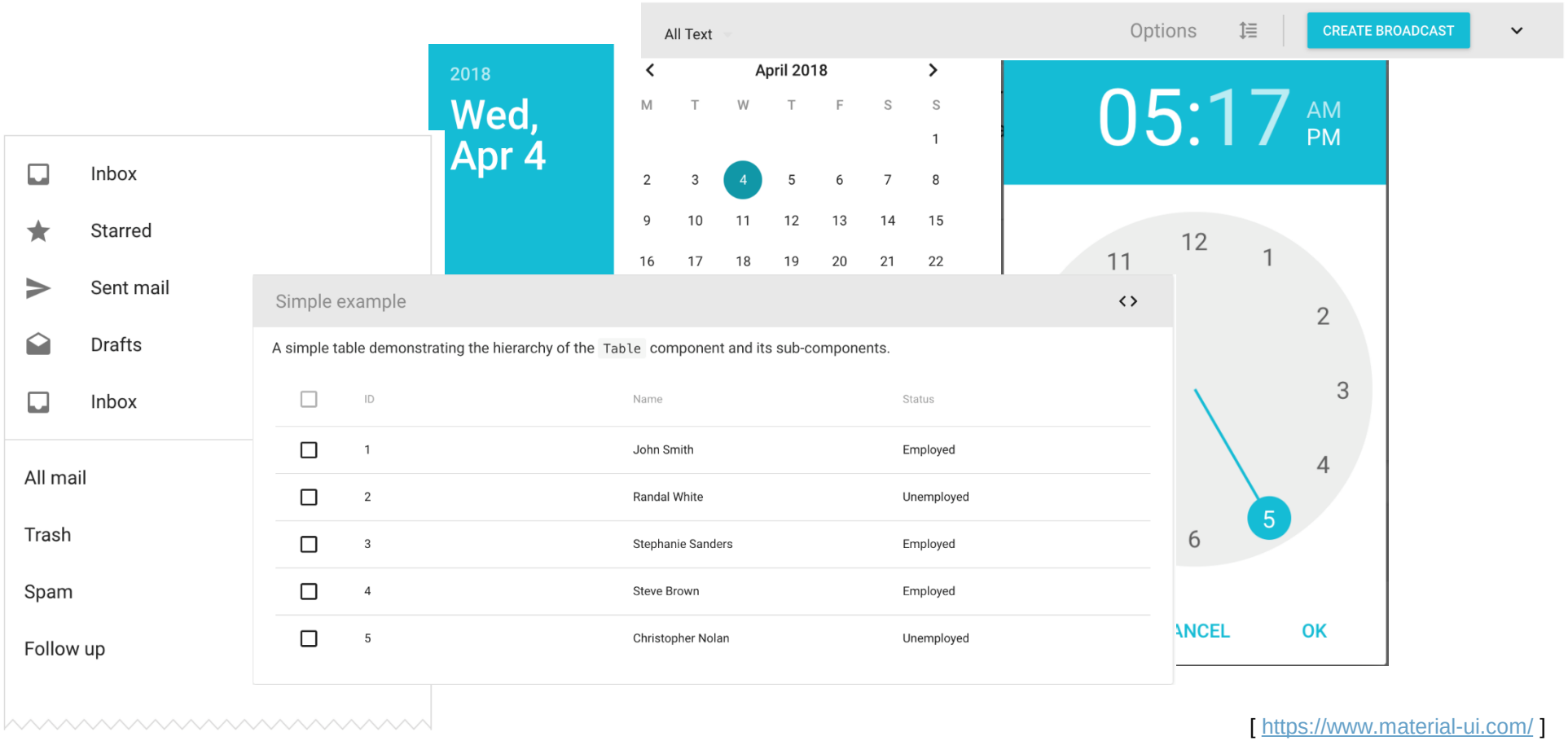
```
<script src="scripts/angular.js" type="text/javascript"></script>
<script src="scripts/angular-material.js" type="text/javascript"></script>

<link href="styles/angular-material.css" rel="stylesheet" type="text/css" />
```

[https://www.material-ui.com/]

Material-UI – Web form controls

- Material-UI provides React components to manage web form controls and their layout, e.g., buttons, date picker, tabs
- Components are organized as independent working components



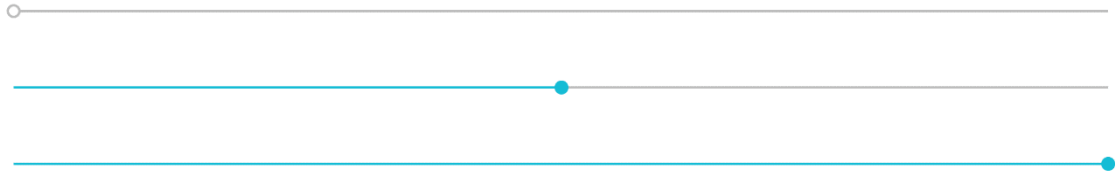
[https://www.material-ui.com/]

Material-UI - Example

Components by Material-UI

```
import React from 'react';
import Slider from 'material-ui/Slider';

/**
 * The `defaultValue` property sets the initial position of the slider.
 * The slider appearance changes when not at the starting position.
 */
const SliderExampleSimple = () => (
  <div>
    <Slider />
    <Slider defaultValue={0.5} />
    <Slider defaultValue={1} />
  </div> );
export default SliderExampleSimple;
```



Outline

4. Developing Serverless Single-Page Web Applications

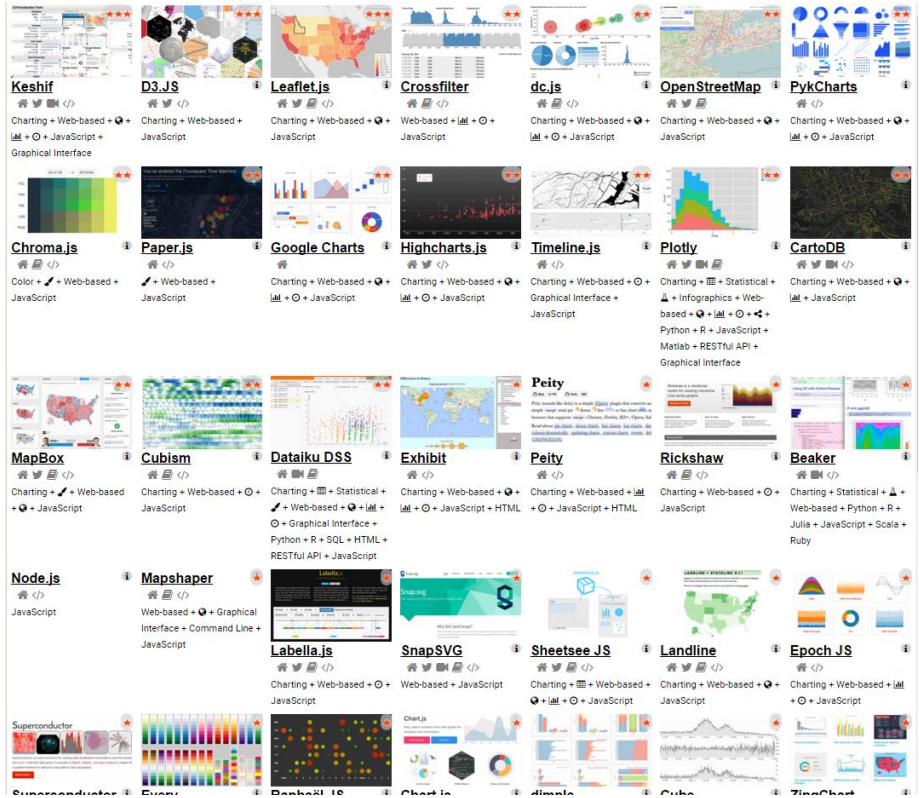
- Introduction: Architecture of Web Applications
- HTML Concepts and Evolution
- CSS Concepts and their Link to HTML
- JavaScript Basics
- Asynchronous operations in JavaScript
- Single-Page Applications
- From Mockups to Web Components
- Component-based Frameworks
- Other web application frameworks
  - Libraries for simplifying content manipulation (jQuery, D3.js)
  - UI design frameworks (Bootstrap, Material Design)
  - Classification of web-based data visualization libraries

Classification of data visualization libraries

There is a huge ecosystem of web-based data visualizations libraries.

Differences regarding:

- **Interactivity**:
  Read-only vs. Manipulable
- **Visualization types**:
  Charts vs. Graphs vs. GeoMaps vs. …
- **Configurability**:
  Programmatic vs. Declarative
- **Technology**:
  SVG vs. HTML5 Canvas vs. …



[http://keshif.me/demo/VisTools]

Programmatic vs. declarative visualization definition

Programmatic Definition

- Visualizations are defined by programming
- Example: (Plain) D3.js

```
var chart = d3.bullet().width(width).height(height);

d3.select("body").selectAll("svg")
    .data(data)
    .enter().append("svg")
    .attr("class", "bullet")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
    .append("g")
    .attr("transform", "translate("
        + margin.left + "," + margin.top + ")")
    .call(chart);
});
```

\+ High customizability

[https://d3js.org]

Declarative Definition

- Declarative configuration of data and visual appearance
- Example: vega

```
{
    "width": 400,
    "height": 200,
    "padding": { "top": 10, "left": 30, "bottom": 30, "right": 10 },
    "data": [ {
    "values": [
        { "x": 1, "y": 28 }, { "x": 2, "y": 55 },
        { "x": 3, "y": 43 }, { "x": 4, "y": 91 },
        { "x": 5, "y": 81 }, { "x": 6, "y": 53 } ] }
    ],
    "axes": [
        { "type": "x", "scale": "x" },
        { "type": "y", "scale": "y" }
    ]
}
```

\+ High reusability

[https://vega.github.io/vega/]

SVG vs. HTML5 Canvas

Scalable Vector Graphics (SVG)

- Vector- and DOM-based
- Defined in XML and appended to the DOM
- Events are also DOM-based

```
<svg>
  <g transform="rotate(45 50 50)">
    <line x1="10" y1="10" x2="85" y2="10"
          style="stroke: #006600;" />
    <rect x="10" y="20" height="50" width="75"
          style="stroke: #006600; fill: #006600" />
    <text x="10" y="90" style="stroke: #660000; fill: #660000">
      Text grouped with shapes
    </text>
  </g>
</svg>
```

+ Facilitates interactivity through manipulation by CSS and JS
+ High fidelity and scalability

[http://tutorials.jenkov.com/svg/svg-element.html]

HTML5 Canvas

- Pixel-based, Bitmap
- Graphics API to draw and modify pixels
- Manually implemented interactions based on mouse coordinates

```
function drawAlaska() {
  var canvas = document.getElementById("myCanvas");
  var ctx = canvas.getContext("2d");
  ctx.beginPath();
  ctx.moveTo(777.5514, 1536.1543);
  ctx.bezierCurveTo(776.4904, 1535.0933, 776.7795, …);
  //
  // 2,875 more path-drawing directives
  //
  ctx.closePath();
  ctx.fillStyle = "#cdc3cc";
  ctx.fill();
}
```

+ High performance, particularly for real-time and high-volume data presentations

[http://www.sitepoint.com/how-to-choose-between-canvas-and-svg/]