Faculty of Science,
Technology
and Medicine

Web Programming

Volker Müller
University of Luxembourg

JSON

JSON = JavaScript Object Notation (RFC 4627)

Lightweight computer data interchange format

Text-based, human-readable

Subset of Javascript

Basic types: Number, String, Boolean, Array, Object, null

JSON Example

```
{
    "firstName": "Volker",
    "lastName": "Müller",
    "address": {
        "streetAddress": "rue XYZ, 12",
        "city": "Luxembourg"
    },
    "phoneNumbers": [ "6534",  "+49 651 12345" ]
}
```

Exactly equal to definition of objects in JS (remember last week)

Using JSON Data on Client Side

Set desired format in request:

```
request.responseType = "json";
```

Assign response to an object, conversion to JS object will be done (almost) automatically:

```
xhr.onreadystatechange = function ( ) {

  if (xhr.readyState == 4 && xhr.status == 200)

     {   object = JSON.parse(xhr.response);   ..... }
```

jQuery (jquery.com)

Fast, relatively small and concise JavaScript Library

Simplifies HTML document traversing, event handling, animating, and Ajax interactions

CSS-like syntax for access of elements in combination with many predefined operations and data structures → special focus on designers

Cross-browser

Some Examples

```
$("div.contentToChange p").size();

$("div.contentToChange
p.firstparagraph:hidden").slideDown("slow");

$("div.contentToChange
p:not(.alert)").append("<strong class='added'> Text
</strong>");

$("div.contentToChange p.par3").hide("slow");
```

Basic Idea of jQuery

Access operator $(...) can be used with HTML tag names, CSS selectors or DOM specifiers, returns list of all elements of that type

Large set of predefined functions available that can be applied to that list

Functions can also be chained since each function returns changed list

Event handlers can be directly defined with JS closures

Example: Simple Sliding Menus

```
$(document).ready( function() {
    $("#drop_down").hide( );
    $("#drop_down").animate({  opacity:0.5  });
  $("a:contains('Google')").click(
            function( ) {  $("#drop_down").show("slow"); });
  $("#drop_down").mouseout (
            function( ){  $(this).hide('slow');  });
});
```

```
<div id="drop_down"> stores
 menu
```

More jQuery Features

jQuery event handlers can be easily written

jQuery has very easy to use Ajax support (for client side)

jQuery UI components exist

Large list of jQuery plugins exist

jQuery also exists in variant for mobile apps

Overview on Newer JS Features

Latest JS versions introduce "common" programming language features like "modules"

HTML5 added many new JS APIs for specific tasks

Tools for efficient development of JS scripts have been developed (similar to PHP or Java)

Web Storage

Application can store data locally within user's browser (not only in cookies as before HTML5)

Amount of stored data can be large (at least 5 MB)

Same origin policy applies – different domains have different stores

Two variants exist: store data with no expiration date, or store until end of session (=browser tab is closed)

Web Storage Example

```
If (typeof(Storage) !== "undefined") {
  localStorage.setItem("lastname", "Mueller");
  let x = localStorage.getItem("name");
```

Similarly for session scope:

```
  sessionStorage.setItem(….)
  sessionStorage.getItem(...)
```

Cache API

Storage mechanism for request-response object pairs

Several (named) cache objects can exist, user responsible for update/delete

Cache size limited, browser may delete (complete) cache object if space needed

Not related with HTTP caching headers

Web Workers

Execution of non-async scripts in a page → page becomes unresponsive until script finished

Web Worker = JavaScript code defined in external file which is running in background, without affecting page performance

Tutorials:

https://www.w3schools.com/html/html5_webworkers.asp

https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

Initializing a Worker

```
if (typeof(Worker) !== "undefined")
{
    w = new Worker("demo_workers.js"); ...
}
else {  // Sorry! No Web Worker support.. }


w.terminate()   // stops a web worker
```

Filename with code for worker provided

Communication Worker – Caller

postMessage() - used to post a message (both caller → worker or worker → caller)

Using posted message:

```
onmessage = function(event) {

 document.getElementById("res").innerHTML =
event.data;  };
```

Data are copied, not shared

Remarks – Shared Workers

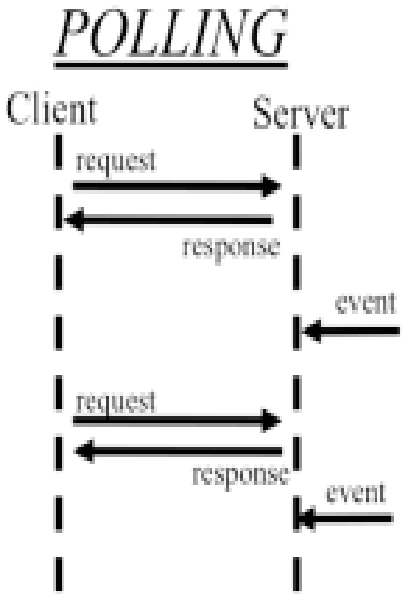
Since web workers are in external files, they do not have access to the build-in JavaScript objects "window" or "document"

If information from these objects should be used, then these infos must be transferred with postMessage from caller to web worker

Shared worker = worker accessible by multiple scripts

Different constructor: new SharedWorker(...)

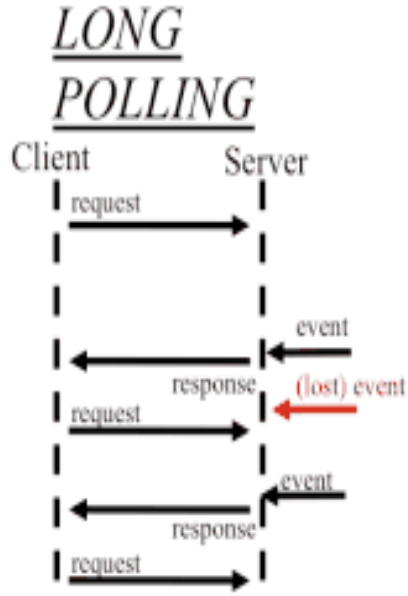
What are Web Sockets?



HTTP is request-response protocol

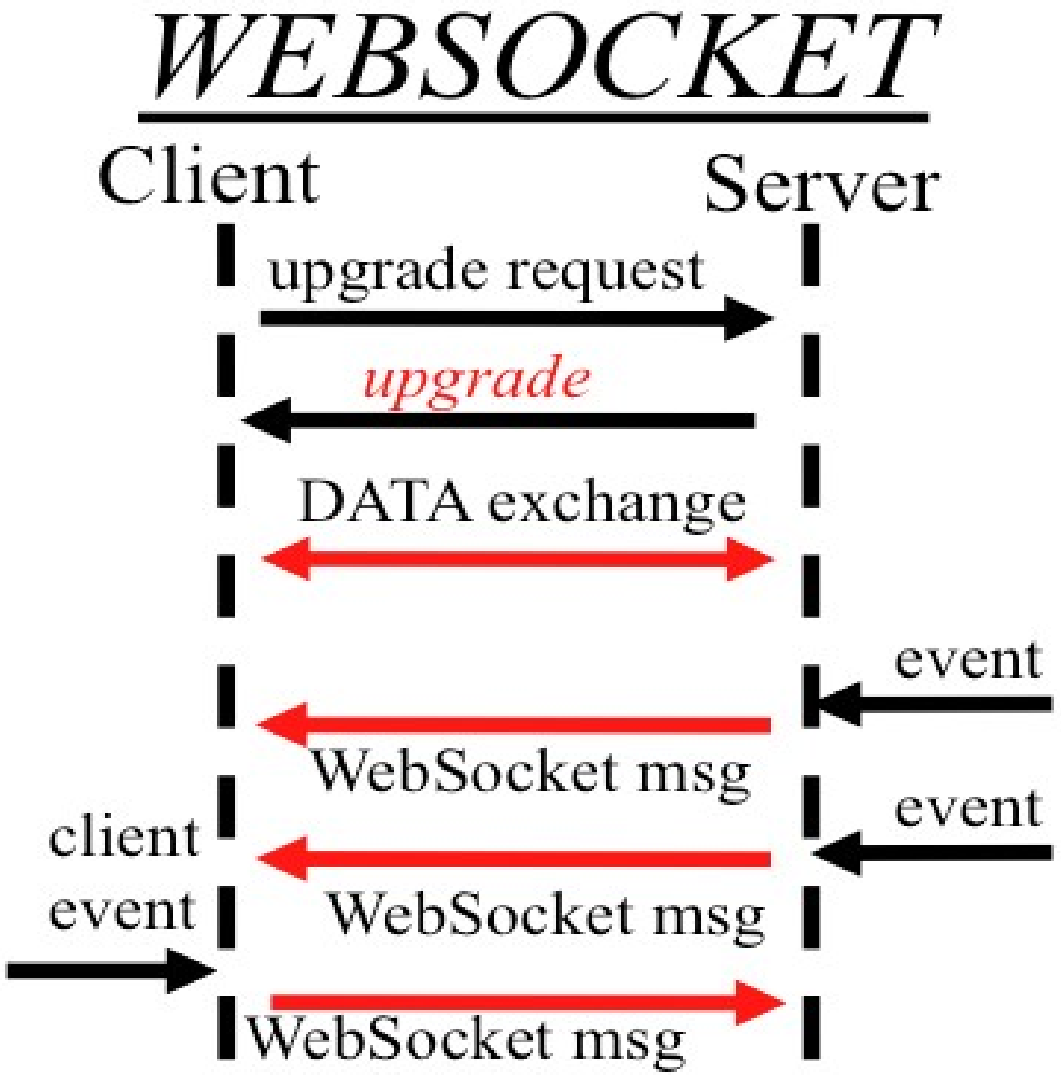
Request initiated by the client

Push data server → client usually uses **polling** (or **long polling**), not very efficient



WebSocket = bidirectional and (full-) duplex communication standard

Communication Flow for WebSockets



Properties of Web Sockets

Independent TCP-based protocol, standardized in RFC 6455

Communications done over TCP port number 80 (or 443) → reuses the normal HTTP(S) ports

Web Socket handshake interpreted by HTTP servers as HTTP Upgrade request

Web Sockets enable stream of messages

Example: WebSocket Client – JS API

```
var mySocket = new WebSocket ("ws://localhost:8080/app/ul");

mySocket.onmessage = function(event)

    { alert("Server message:" + event.data); };

mySocket.onopen = function(event) {...};

mySocket.onclose = function(event) {...};

mySocket.onerror = function(event) {...};

mySocket.send("Hallo Server!");

mySocket.close();
```

WebSocket Server with PHP

Ratchet = loosely coupled PHP library providing developers with tools to create real time, bi-directional applications over WebSockets

Based on the PHP package manager composer

Defines a PHP Object for server side, which is then instantiated

Code and docs available at https://socketo.me

```
use Ratchet\MessageComponentInterface;
use Ratchet\ConnectionInterface;


class Chat implements MessageComponentInterface {
    public function onOpen(ConnectionInterface $conn) { … }
    public function onMessage(ConnectionInterface $from, $msg)
    { … }
    public function onClose(ConnectionInterface $conn) { … }
    public function onError(ConnectionInterface $conn, Exception
$e) { … }
}
```

Run the Server

```
use Ratchet\Server\IoServer;

use MyApp\Chat;


require dirname(__DIR__) . '/vendor/autoload.php';

$server = IoServer::factory(new Chat( ), 8080);

$server->run();
```

Conclusion WebSockets

WebSockets are new technology which becomes increasingly popular

Improves efficiency for all application with a “push” operation / data streaming

In new version of HTTP (HTTP/2), similar push functionality already build into protocol, but for server push-only

Main New Features in HTTP/2

Binary protocol (whereas HTTP 1.x is textual)

HTTP/2 is multiplexed (HTTP 1.1 pipelining uses FIFO → slow request can block connection)

HTTP/2 uses header compression to reduce overhead

HTTP/2 allows server push to proactively send data into client cache

Modules in JS

Modules allows separation of code parts into different files, where part of the code can be hidden from the outside

Exported functions / objects / variables of module can be used by other programs (with an import), non-exported parts are not visible outside module

Allows packaging of code into different files

Module Example – lib.mjs

```
export const foo = Math.sqrt(2)

export function cube(x) { return x*x*x; }

export class A { … }
```

Default export can be defined

Often extension ".mjs" used for files with JS modules

Module Example – Import into JS

```
import * as myModule from './lib.mjs'

import {cube as c} from './lib.mjs'
```

For now: ./ needed, might be changed in future ("./" = site root)

Usage:

```
myModule.cube(2);

c(2);
```

"Import" into HTML: `<script type="module" src="…">`

Difference Module Code – "Normal" Code

Modules have "strict mode" enabled per default

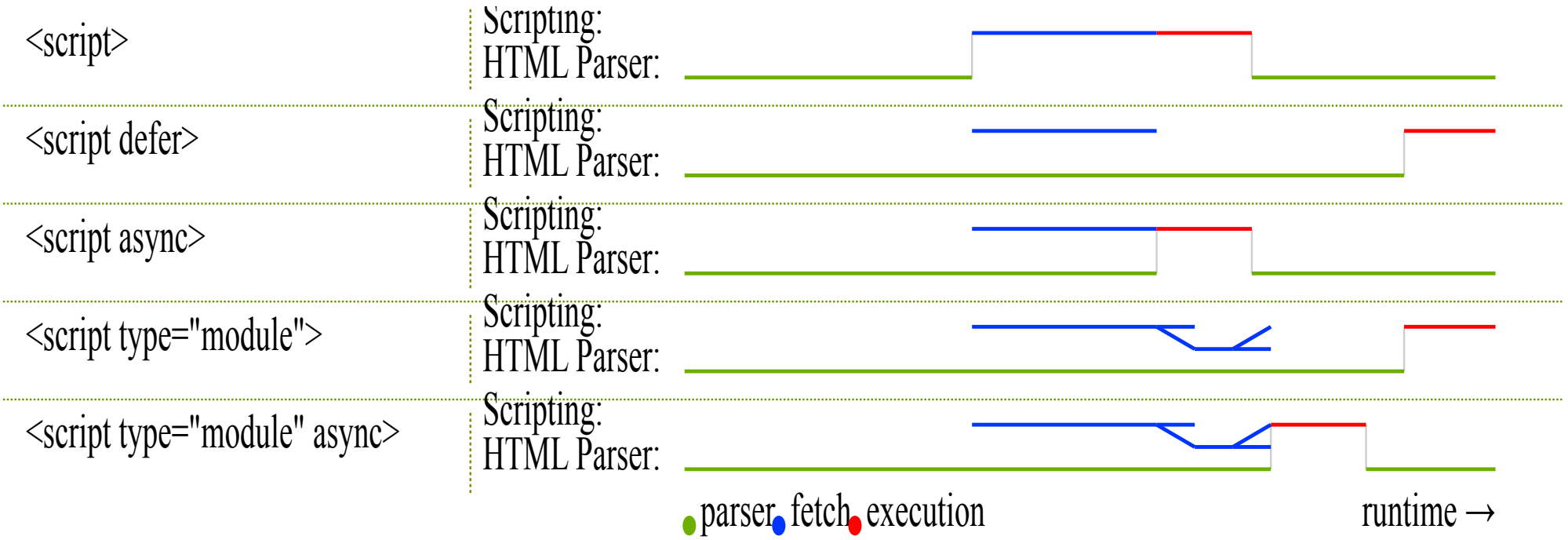
Strict mode replaces silent errors to "throwing errors", fixes mistakes, prohibits some syntax likely to disappear

HTML syntax (comments) not allowed within module

var in module does not create global variable

→ JS code outside module might behave different from same code inside module

Modules are per Default "Deferred"



More information: v8.dev/features/modules

npm (www.npmjs.com)

npm = package manager for JS, written in JS

Default package manager for Node.JS

Manages also dependencies (similar to composer)

Searchable central repository with JS modules publicly accessible at npmjs.com

Global mod. installation: npm install -g <modname>

Local package installation done with configuration file package.json (created with npm init)

Example: package.json

```
{
  "name": "project",     "version": "1.0.0",   "description": "",
  "main": "index.js",
  "scripts":   {  "start": "node index.js"  },
  "dependencies": {
     "redis" : "^2.8.0",
     "os" : "0.1.1"
   },
}
```

"npm install" will read file and install all needed modules

Recommendation: have a look at the provided Docker example!

Scopes in NPM

NPM package names can contain scope

Example: @somescope/packagename

Scopes provide way to group related packages together

Equal package names possible if different scopes

Each npm user/organization can get own scope -→ signal official packages for organizations

Alternative: Yarn

Many other dependency managers for JS exist

Yarn (yarnpkg.com/): developed by Facebook

- Uses more local caching
- Parallel operations → generally faster compared to npm
- Deterministic dependency resolution
- Usually uses same repository as npm

Build System - Gulp

Gulp = popular build system developed in JS

Used for automation of time-consuming and repetitive tasks like code minification, concatenation, translation TS → JS, unit testing, linting, …

Based on node.js

Tasks defined in configuration "gulpfile.js"

Example Gulpfile

```
var gulp = require('gulp'),
 minifyCss = require("gulp-minify-css");


gulp.task('minify-css', function () {
gulp.src('./CSS/one.css')
.pipe(minifyCss())
.pipe(gulp.dest('path/to/destination'));
});
```

require = module handling in node.js

gulp-minify-css = predefined gulp module, many more exist

Run it: gulp minify-css

TypeScript

TypeScript is a strongly typed superset of JS (ES6 and newer), which compiles to plain JS

Developed by Microsoft (www.typescriptlang.org)

TypeScript can not be run directly in browser, but has to be compiled to JS to run

Installation: npm install -g typescript

Translation with tsc integrated in build process (mvn, gradle, gulp)

Example TypeScript (Extension .ts)

```
interface Person { firstName: string; lastName:
string;}

function greeter(p: Person)

  { return "Hello " + p.firstName; }


let x = "Volker";

let y = greeter(x);   // produces error
```

Advantages

Supports static typing, strong typing

Supports classes and interfaces

Provides error-checking at compile time

Provides many (already existing) definition files for external JS libraries

→ Speeds up development time, since errors (especially wrong types) can be found earlier

Next Week

JS on the Server Side: Node.JS