

1. pt.

NSWI 142

Základy Weby

Client

+ Browser

↳

http:// ... / cs.php

↳ port 80

DNS server

↓

195.113.20

TCP pripojení

server

(Apache)

URI

(IRI)

<schema>: <hierarchical part> ? <Query> # fragment

(optional)

URL

- URL popisuje lokaci nejakeho zdroje

http://webi.../vJU2ASHm-1

HTTP request

- methods:

GET (data dostat z serveru)

POST (poslat data serveru)

HEAD (pouze response headers)

PUT/DELETE (special)

- request URL

- ↳ cesta (třeba s query) URL
- ↳ specifikace toho, co chci  
(filtér na e-shopu)

## HTTP responses

- 1xx informace /
- 2xx úspěch  
200 ok, 204 no content
- 3xx potřeba akce  
(301 permanentní přesun)
- 4xx client side error  
(404)
- 5xx server side error

## HTTP odesílání /

- ↳ stažování obsahu ze serveru
- ↳ upload mezipříruček na server

- problémy:

- bez základní komunikace  
(bez kontextu, stateless)
- iniciativní pouze klientem  
(zdáreň update)

↳ RESTful 2/3 (multiplex)

## Web of documents

- hyperlinky ~ URL

# HTML | DOM ~ struktura

Document  
Object  
Model

```
<!DOCTYPE html>  
<html>
```

...

```
<body>
```

```
<h1>DOM Ex. </h1>
```

```
<p>
```

Document

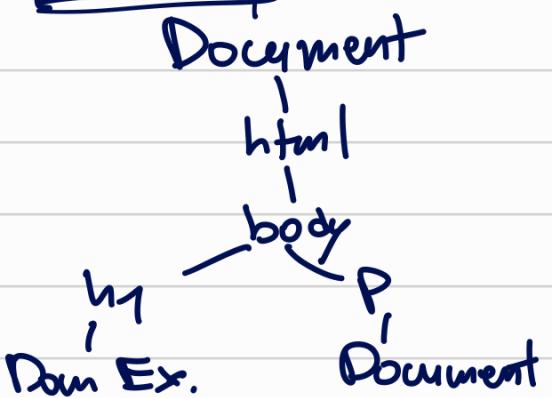
```
</p>
```

...

```
</body>
```

```
</html>
```

Dom:



## Elements HTML

- např. form ~ formulář, post/get

## CSS (stylu HTML)

- cascading style sheets
- do uzelů DOM sbíruji daje vlastnosti
- základní      body → h1
- vnitřní/externí      } (tady aplíkuji pro body, bude platit i pro h1, pokud nějž má)
- (ale odděleny/soubor nebo umístěn HTML)



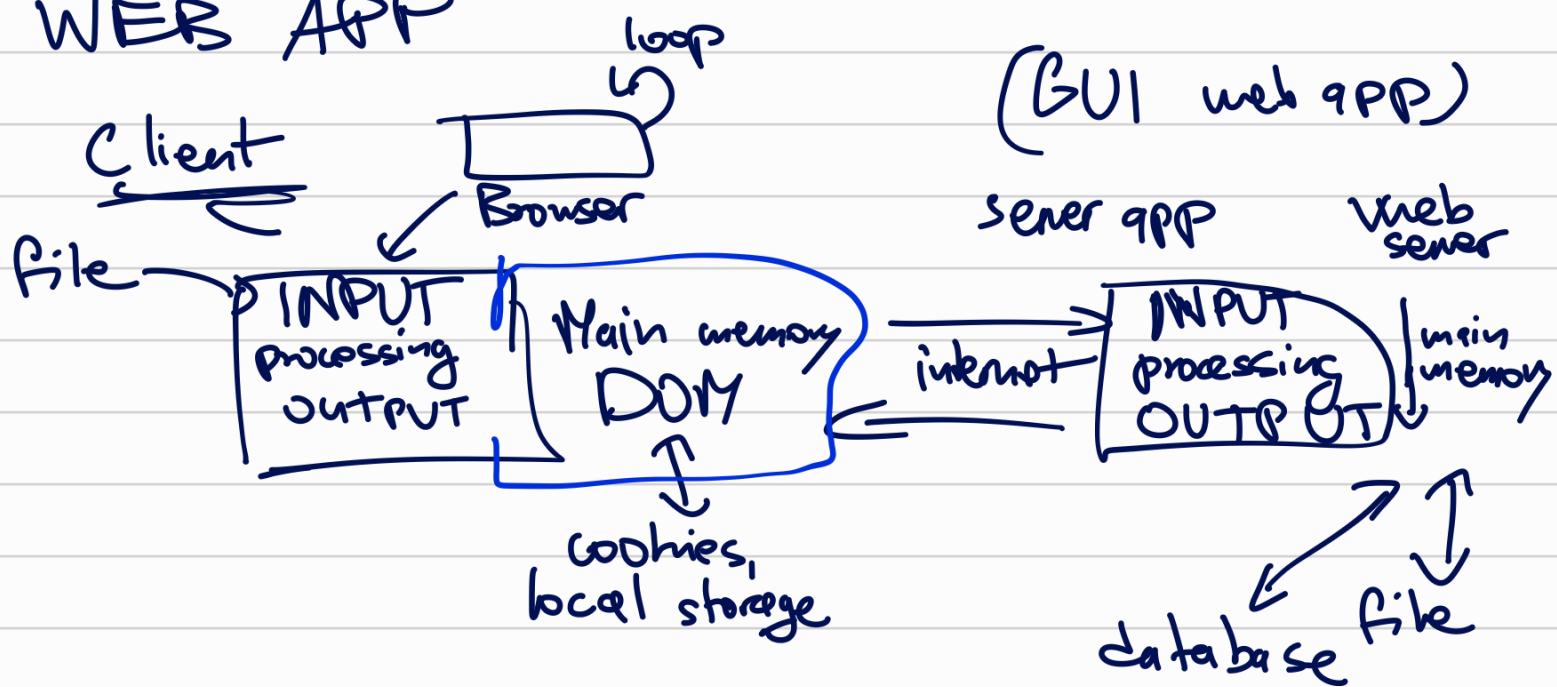
umístění: static/relative/...

# CSS pre-processing

## SASS

- pořízení proměnných  
(za komplikaci, ne runtime)

## WEB APP

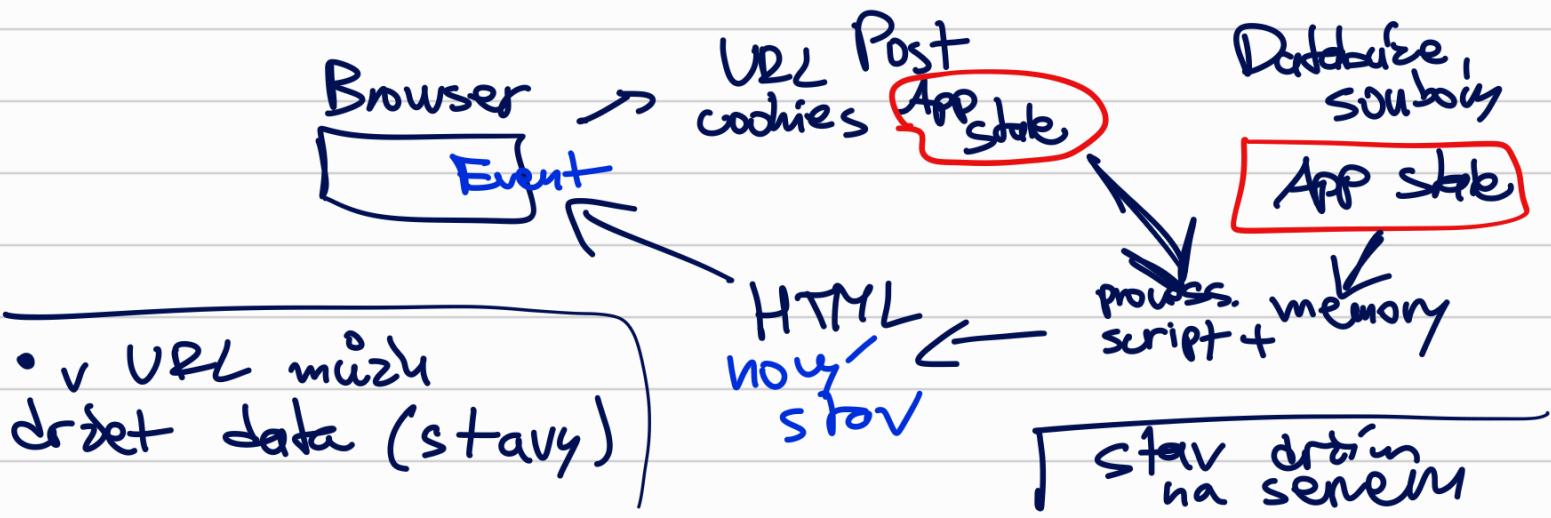


## Single-page app

- file je na straně serveru,  
AJAX requesty

## App state (stav)

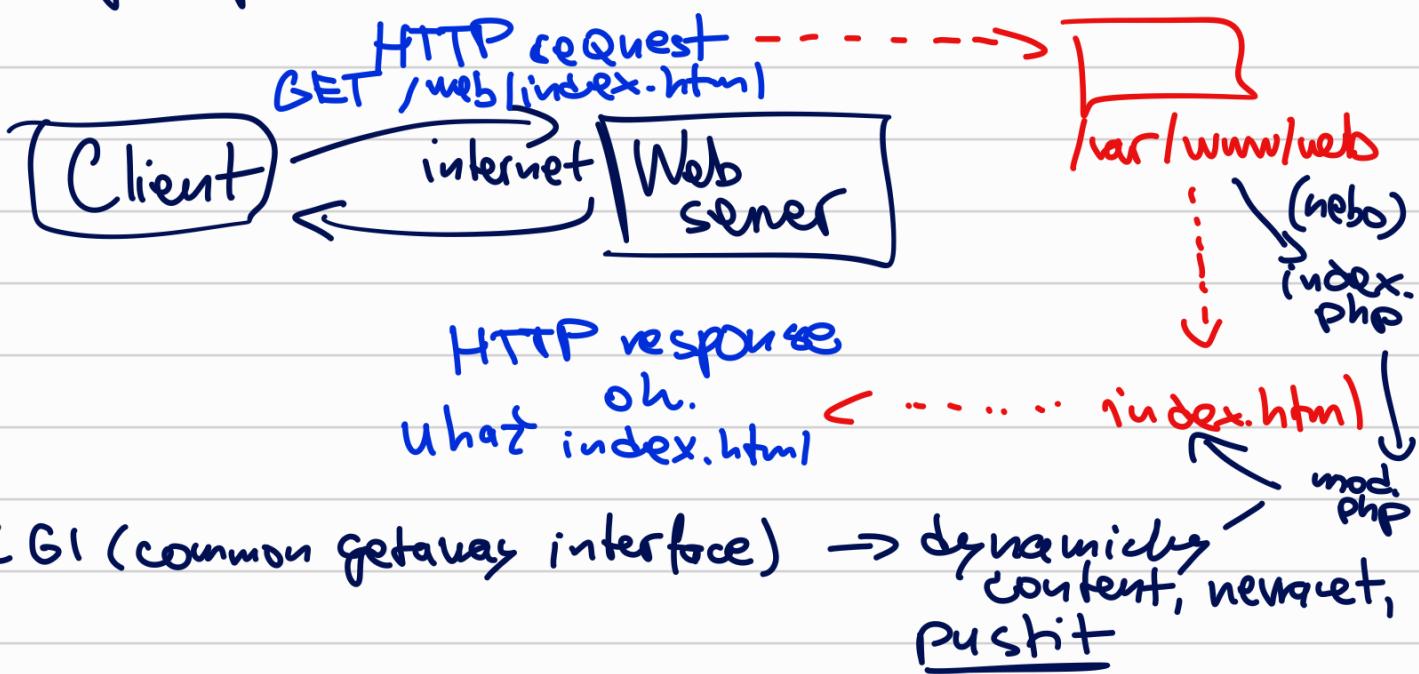
↳ např. že se refreshem vymaze form  
↳ či přihlášení (URL, cookies,...)



• to nájdete  
druhé URL

# Web server

- host. posílání emailu



Jiné jazyky ve weboch:

Python → Flask, Django

C# → ASP. NET

Java → Java Server Pages

Node.js → JS server-side platform

## Weak Dynamic Typing

- PHP, SS, Python
- Type uložen ve values (poměrně bez typu)
- runtime
- poměrně jen registrace jinčí

\$x = 42;  
\$x = "hi"  
OK  
routine check

function foo (int \$count)

## Strong Static T.

- C#, C++, C, Java
- Type uložen v proměnných a values
- compile
- poměrně v paměti

int x = 42;  
x = "hi" Error

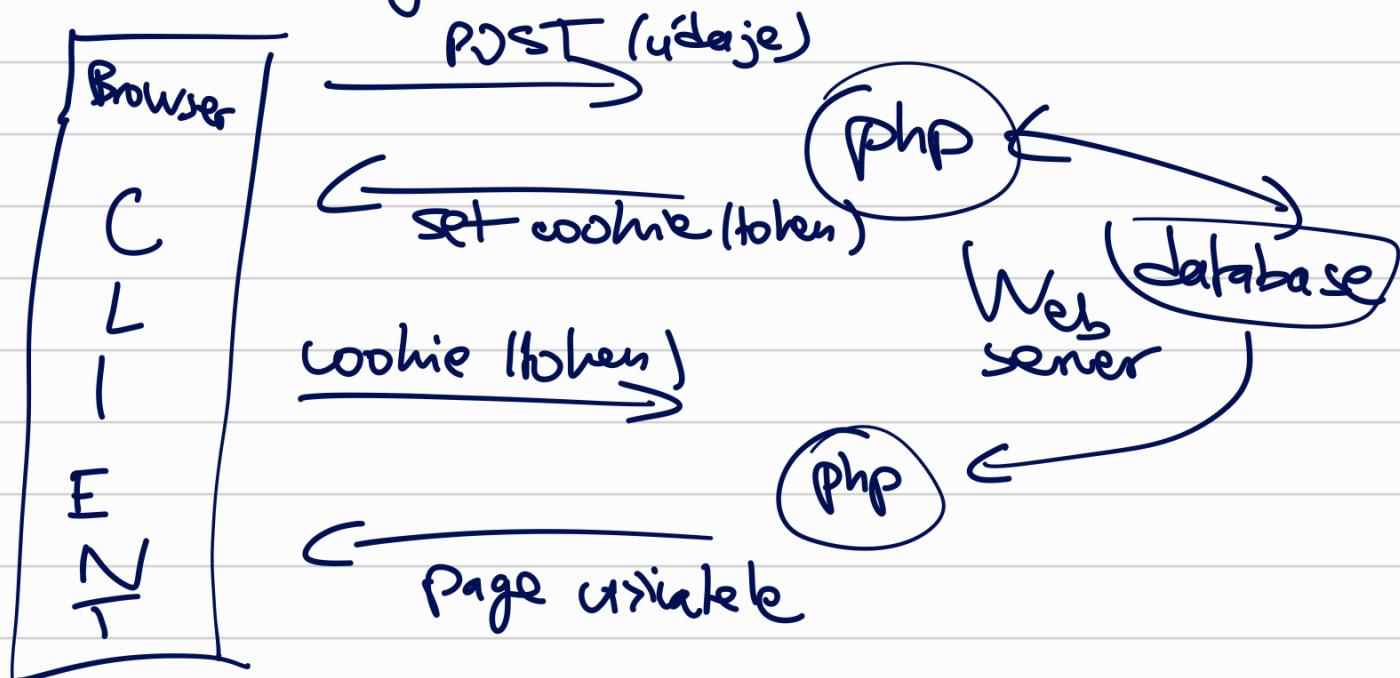
auto y = 54; compiler

# PHP

- zadáva's <?php
- value mají explicitní typ
  - ↳ scalar (bool, int, float, string)
  - ↳ compound (array, object)
  - ↳ special (resource, NULL)
- funkce vystavěné - is\_int() ...
- proměnné: \$var1  
(není potreba deklarovat)

\$a = 'b'; \$<sup>b</sup> \$a = 42  
• stejný jmeno b = 42

## Session management



## PHP Front Controller

HTTP → index.php  
product.php

Request, render  
response  
autentification  
sanitization vs input...

# Routing and dispatching

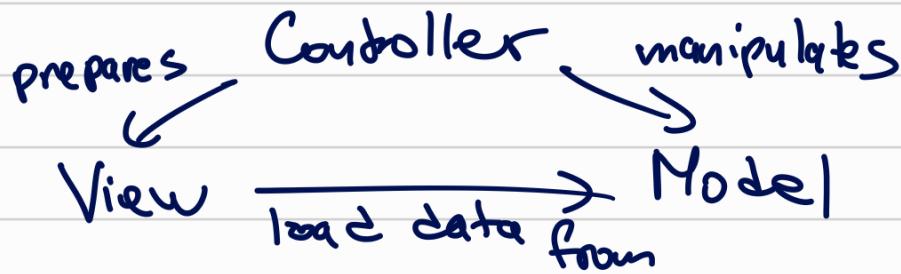
- initialize libraries

FC  specifies frontoller — Ahce  
routing / dispatching

Sablony PHP ~ fajn

# Design pattern, API

↓  
MVC → model, view, controller  
API      UI              Business logic



## View

- šablonovalní ~ HTML oddíly a funkce a t.d.  
(generování headers a footers)

## Controller

- GET: POST requesty řeší

Komponenty ~ díl se „schematicky“ rozdělit  
o s vnitřní komunikací a t.d.

- software modul, o něž umožňuje  
(funkcionalita)

## Container dependency

### Dependency injection

- hledá komponentu si řešíme, jakej jiné komponenty potřebuje
- externí injekce
- deklarace potřebujících dependencies

příklad:

„ja'chá komponentu, než do provozu“

/\*\*

\* @ Component WePage

\*/  
class WePageController implements IController  
{  
 /\*\* @ inject IDatabase \*/  
 public \$db;  
 inject by interface

/\*\* @ inject name="NewService" \*/  
 public \$news;  
 inject by name

function \_\_construct (ILogger \$log) { ... }  
}

\* position event

function process (\$comment) {  
 ...  
}

but predefine do factory a) rebo set,  
rebo prime  
do method

class a {

private \$logger;

public function \_\_construct (Logger \$logger) {

\$this->logger = \$logger;

\$logger = new logger();

\$users = new q(\$logger);

}

# REST (Representational state transfer API)

- HTTP
- hledá věc podle URL
  - pro editaci GET, PUT, POST, DELETE
- client-server, bez stavu
- cachování
- interface, ke kterému se chováme stejně
- vrstvený systém (proxy, ...)

příklad:

GET	/galerie	/galerie/cats	/galerie/cats/cat01
	list všech galerií JSON	list cats galerie JSON	první fotka JPEG

## ORM ~ Object-relation mapping

- na synchronizaci dat s databází
- užívá definovaný datový model a pomocí dodatečné informace ho mapuje do databáze
  - ↳ funkcionální framework zajišťující průměrnost
- i je schopné projet prázdnou databázi a vytvořit tabulky (i migrace)

# Javascript

↳ client-side scripting

- slabé typovanie /

↳ type of (holí se)

- automatically GC

- converse automaticke'

$$\begin{array}{l} "5" + 4 \rightarrow \underline{\underline{54}} \\ "5" \cdot 4 \rightarrow \underline{\underline{20}} \end{array}$$

- premenne /

do pridanie jednej konst count = 1;  
jeu jednoj let count = 1;  
pričiaklo ->

(dôle)

var a nice!

- first-class functions

```
<script type="text/javascript"></script>
```

```
<script type="text/javascript" src="url"></script>
```

DOM levels Level 0 - sa funkce

Level 1 - navigace

2 - eventy, CSS

3 - keyboard event

4 - výrob (nem' potřeba)

Event model ~> se hdy stane

- handling ~> target element DOM

- event queue

(funkce po sobě, 1 vrchno)

globální premenne / window.a = 123

var promenne' ~ u cele' telle funkci'  
vistelne'  
~> po hude deklaracne' mimo  
funkci = globální

visteknost  
↳ postupna'

function f1() {  
let a2;

f. f2() {  
let a3;

f. f3() {  
vidr a2 i a3

}}}

Prototypes ("classes")

const S = {  
"side": 0,  
"p": function() {  
return 4 \* this.side;  
}},  
S

const tile = Object.create(Square);

tile.side = 42;  
console.log(tile.p());

This • keyword

↳ global - script env. (window)  
↳ function - zakres na funkci

class (ano, vč JS 6)

• getter / setter, ... (viz NSWI 153)

## Promises

↳ completable / selhalný asynchronních akcí  
• then, • catch, • finally  
• uspěch fail dodělá se  
(u/f.)

Fetch(url).then(response => ... )

State synchronizace (State v DOM)

• data se uloží do DOM i JS memory

AJAX a UI možnosti

1) full reload

↳ po úspěšné requestu JS reloaduje stránku

2) Císteče / optimalizované řešení

↳ po úspěšné requestu JS spustí reload na danou komponentu přes my /  
AJAX GET request

3) opt. responder

↳ POST odpovídá pořídku HTML  
fragment nebo data komponent  
pro pre-renderování / fabulky

jQuery  $\rightsquigarrow$  JS knihovna

- zjednodušená DOM manipulace
- Event handling

React  $\rightsquigarrow$  upravuje jen rozdíly v DOM  
 $\rightsquigarrow$  Virtual DOM

Angular.js  $\rightsquigarrow$  staticky přepisuje (pouze)