# JavaScript
# Web API, Vue.js úvod

kurz Webové technológie

Eduard Kuric

STU FIIT

1. 12. 2020

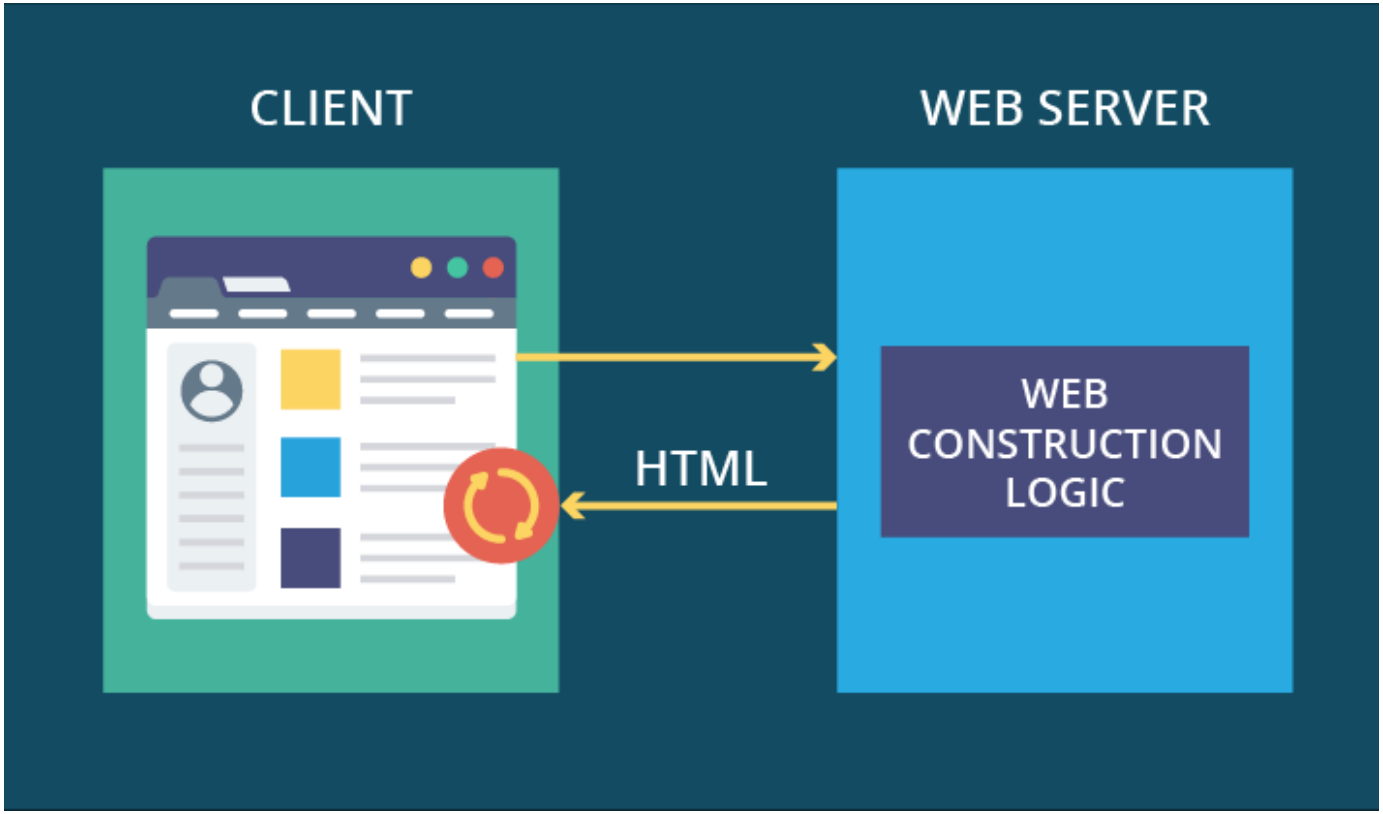
- Web API
  - Získavanie údajov zo servera (AJAX, Fetch API)
    - JSON, Promises
  - Ukladanie údajov na strane klienta (sessionStorage, localStorage, IndexedDB)
- Moduly
  - AMD (RequireJS), CommonJS (SystemJS), ES6
- Transpilátor (transpiler - Babel)
- Balíkovač (bundler - Browserify)
- Minifikácia, zamlženie kódu (obfuscation)
- Webpack
- JS režim strict
- Vue.js úvod
  - MVVM
  - Reaktivita
  - Komponenty
  - Základné direktívy

# JavaScript WEB API

# API prehliadačov

- Manipulácia s dokumentmi (DOM)
- Získavanie údajov zo servera (AJAX, Fetch API)
- Ukladanie údajov na strane klienta (Web Storage, IndexedDB)
- Vykreslenie a tvorbu grafiky (Canvas, WebGL)
- Audio a Video (HTMLMediaElement, Web Audio API, WebRTC)
- Prístup k zariadeniam (Geolocation, Notifications, Vibration API)

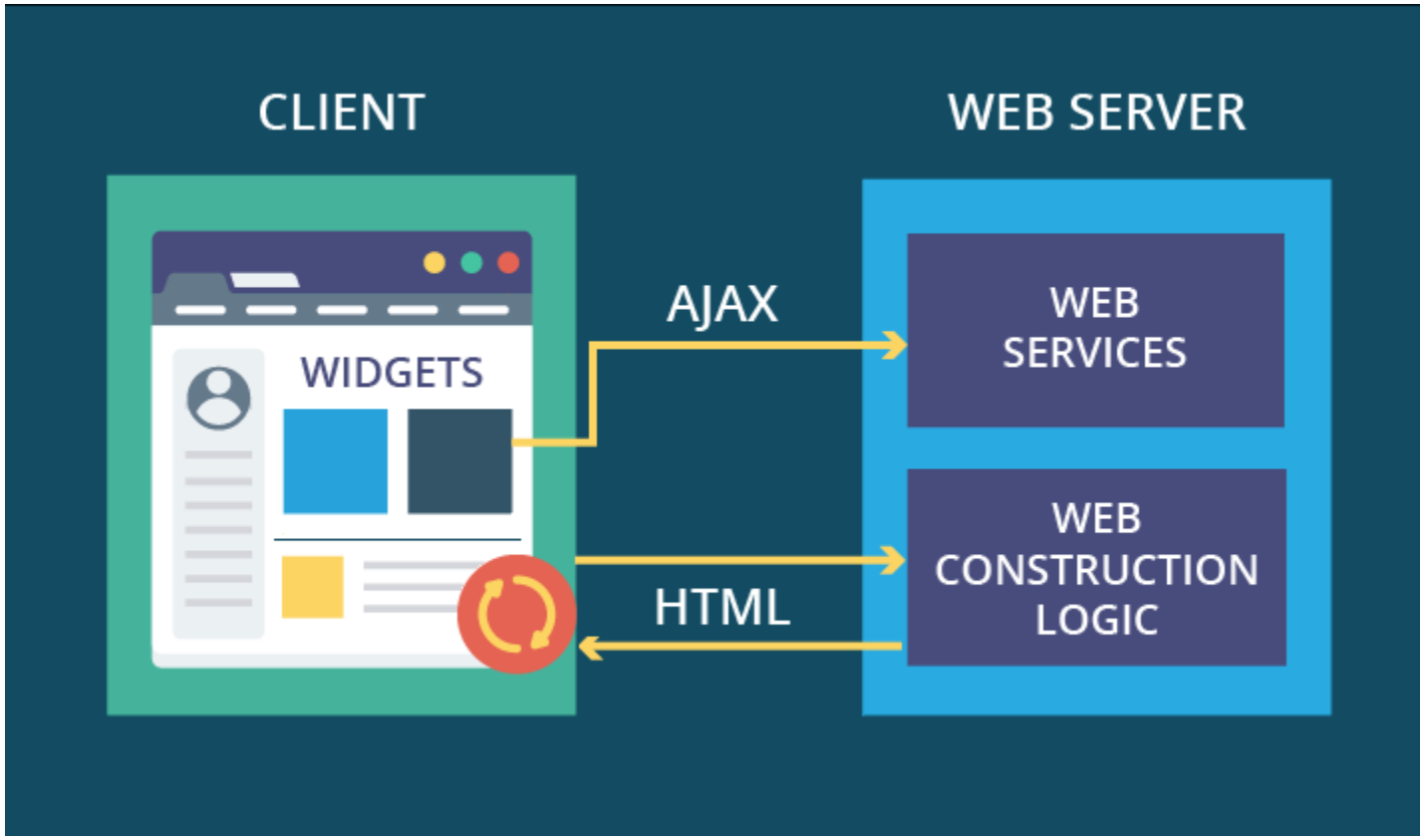
# SSR – Server-side rendering



# AJAX

- Asynchronous JavaScript + XML
- umožňuje meniť obsah stránok bez potreby znovunačítania celej stránky zo servera
- dynamicky načítavané vybrané fragmenty/oblasti stránok cez JavaScript (tzv. JS widgets)
- **obsah fragmentov je napĺňaný cez AJAX query**
  - klient vytvorí (pre fragment) AJAX požiadavku a odošle ju serveru - službe
  - server - logika služby - požiadavku spracuje, vygeneruje údaje a odošle ich klientovi
  - **klient prijatými údajmi naplní fragment stránky**
    - prijatý obsah môže byť HTML, JSON, XML

# JS widgets


CLIENT
WEB SERVER
WIDGETS
AJAX
WEB SERVICES
WEB CONSTRUCTION LOGIC
HTML

# JSON – formát údajov

- JavaScript Object Notation
- textový formát údajov
- vhodný na zápis (krátkych) štruktúrovaných údajov **vymienaných medzi webovými aplikáciami**
- syntax je platným zápisom jazyka JavaScript
  - z JSON údajov môžeme rovno vytvoriť JS objekt

```
{
 "name": "Peter",
 "age": 30,
 "car": null
}
```

# JSON – typy údajov

- `JSONString` – textový reťazec
- `JSONNumber` – číslo (celočíselné alebo reálné, vrátane zápisu s exponentom)
- `JSONBoolean` – logická hodnota
- `JSONNull` – hodnota null
- `JSONArray` – pole
- `JSONObject`– objekt

# Práca so zložitejšími JSON súbormi

- JSONLint
  - validátor
- JSON Editor
  - stromová štruktúra
  - vyhľadávanie v údajoch

# XML vs JSON

- Obidva sú hierarchické
- JSON kratší, rýchlejší na načítanie/zápis
  - JavaScript objekt
- XML – potrebujeme parser
- JSON – JS funkcia
  - JSON.parse(‘’)

# XMLHttpRequest

- často skrátene XHR

```
var request = new XMLHttpRequest();
request.open('GET', url);
// document, json, blob
request.responseType = 'text';
request.onload = function() {
   ...
};
request.send();
```

# XMLHttpRequest /2

- asynchrónna operácia, musíme počkať na dokončenie operácie (na odpoveď `response`)
- dokončenie operácie indikuje `load` udalosť, ktorá vyvolá `onload` obsluhu, v ktorej je možné spracovať prijatý `response`

- UKÁŽKA

# Fetch API

- založený na tzv. *Promises* (od ES2015/ES6)

```
fetch(url).then(function(response) {
  response.json().then(function(myPoem) {
    poemDisplayTitle.innerHTML = myPoem.title;
    poemDisplayText.innerHTML = myPoem.text;
  });
});
```

- UKÁŽKA

# XHR vs. Fetch API

- ktorý použiť? závisí aj na projekte...

- XHR má dobrú podporu aj v starších prehliadačoch
  - pomerne stará technológia - koncom 90. rokov, uviedol Microsoft
  - ak má byť projekt prístupný aj v starých prehliadačoch
- Fetch API je modernejšie, založené na *Promises*
  - problém s IE, ale IE sa už ďalej nevyvíja, nástupca EDGE

# Axios vs. Fetch API

- nie sú chyby ako chyby
- Fetch API odmietne sľub iba v prípade sieťových chýb
  - napr. nepodarila sa preložiť adresa, server je nedostupný, CORS nie je povolený
- ak server vráti 404, Fetch API hlási úspech, a teda vykoná `then`, a nie `catch`
- Axios vykoná `catch`

# Promises

- predstavte si, že ste dieťa a mama Vám sľúbila, že Vám kúpi vláčik, keď budete poslúchať (neurčito)
- takýto sľub, alebo promise v JS môže mať tieto stavy:
  - *pending* – čakáte, neviete, kedy ho pôjde kúpiť
  - *resolved* – mama sa rozhodla kúpiť vláčik
  - *rejected* – mama sa rozhodla nekúpiť vláčik, lebo nie je šťastná, neposlúchate
- pretransformujme túto úlohu do JS ...

# Promises – definovanie sľubu

```
var isMomHappy = true;

var willIGetNewTrain = new Promise(
    function (resolve, reject) {
        if (isMomHappy) {
            var train = {
             color: 'čierny'
            };
            resolve(train); // fulfilled
        } else {
            var reason = new Error('Mama nie je šťastná');
            reject(reason); // reject
        }
    }
);
```

```
// Promise syntax
new Promise(function (resolve, reject) { ... } );
```

# Promises – čakateľ na sľub

```
var askMom = function () {
    willIGetNewTrain
        .then(function (fulfilled) {
          console.log(fulfilled);
        })
        .catch(function (error) {
          console.log(error.message);
        });
};

askMom();
```

# Reťazenie sľubov

- pochválite sa kamarátovi

```
var showOff = function (train) {
 return new Promise(
    function (resolve, reject) {
       var message = 'Ahoj, mám nový ' +
                     train.color + 'vláčik';
       resolve(message);
    }
  );
};
```

# Reťazenie sľubov /2

```
var askMom = function () {
    willIGetNewTrain
    .then(showOff)
    .then(function (fulfilled) {
       // Ahoj, mám nový čierny vláčik
       console.log(fulfilled);
     })
    .catch(function (error) {
          console.log(error.message);
    });
};
```

# Sľuby sú asynchrónne

```
var askMom = function () {
    console.log('predtým ako sa opýtam mamy');

    willIGetNewTrain
    .then(showOff)
    .then(function (fulfilled) {
          // Ahoj, mám nový čierny vláčik
          console.log(fulfilled);
        })
    .catch(function (error) {
                console.log(error.message);
    });

    console.log('po otázke');
};
```

# Sľuby sú asynchrónne /2

- očakávali by sme
  - **predtým ako sa opýtam mamy**
  - `Ahoj, mám nový čierny vláčik`
  - **po otázke**
- v skutočnosti môže byť aktuálne poradie:
  - **predtým ako sa opýtam mamy**
  - **po otázke**
  - `Ahoj, mám nový čierny vláčik`

# Život pred sľubmi...

```
// pouzivali sme jQuery a callback funkciu
function addAsync (num1, num2, callback) {
    return $.getJSON('http://myService', {
        num1: num1,
        num2: num2
    }, callback);
}

addAsync(1, 2, success => {
    const result = success;
});
```

# Život pred sľubmi... čo ak?

```
resultA = add(1, 2);
resultB = add(resultA, 3);
resultC = add(resultB, 4);

console.log('total' + resultC);
```

# Život pred sľubmi... takto ...

```
function addAsync (num1, num2, callback) {
    return $.getJSON('http://myService', {
        num1: num1,
        num2: num2
    }, callback);
}
addAsync(1, 2, success => {
    resultA = success;
    addAsync(resultA, 3, success => {
        resultB = success;
        addAsync(resultB, 4, success => {
            resultC = success;
            console.log('total' + resultC);
         });
    });
});
```

# Promises - podpora

- od **ES6 / ES2015**
  - moderné prehliadače natívne
- inak
  - Bluebird - knižnica na podporu JS promises

# Ukladanie údajov na strane klienta

# Web storage APIs

- **sessionStorage** – údaje dostupné iba počas prezerania stránky, key-value (reťazce)
  - otvorenie stránky v novej karte, alebo okne inicializuje nové sedenie (session), reload stránky nemá vplyv
- **localStorage** – údaje sú dostupné aj po zatvorení a znovuotvorení prehliadača; key-value (reťazce)
- **indexedDB** – ukladanie štruktúrovaných dát (aj súbory)
  - použitie indexov na efektívne vyhľadávanie v údajoch
  - životnosť údajov ako localStorage

# Web storage APIs - limit

- localStorage, sessionStorage – 5MiB
  - test, koľko daný prehliadač umožní uložiť
- IndexedDB – minimum (soft limit) – 5MiB
  - maximum – kapacita disku, ale
  - prehliadač požiada o povolenie na uloženie väčšieho množstva údajov

# `sessionStorage`, `localStorage`

- vloženie/uloženie údajov do úložiska
  - `localStorage.setItem('name','Peter');`
- získanie/načítanie údajov
  - `localStorage.getItem('name');`
- vymazanie údajov
  - `localStorage.removeItem('name');`

- **pre každú doménu je vyhradené samostatné/vlastné úložisko**

# IndexedDB (IDB)

- transakčný DB systém – objektový
- tabuľky sú objekty – object store
  - nemajú fixnú schému, je možné ju za behu meniť
- umožňuje okrem reťazcov uložiť prakticky akýkoľvek typ údajov – bloby (obrázky, videá, audio, ...)
- použitie je o niečo zložitejšie

- podpora v prehliadačoch

# Pripojenie/vytvorenie databázy

- metóda open zabezpečí otvorenie pripojenia na danú DB v prípade, ak existuje; inak ju vytvorí
  - druhým argumentom je verzia DB

```
var request =
      window.indexedDB.open('notes', 1);
```

- operácie sú asynchrónne

# Definovanie obsluhy

- onerror

```
request.onerror = function() {
  console.log('Database failed to open');
};
```

- onsuccess

```
let db; // objekt na otvorenu DB
request.onsuccess = function() {
  console.log('Database opened successfully');
  db = request.result;
};
```

# Štruktúra objektu – note

```
{
  title: "Kúpiť mlieko",
  body: "Aj sojové aj kravské mlieko",
  date: "2012-04-23T18:25:43.511Z"
}
```

# Definovanie/štruktúra databázy

- definovanie, alebo zmena štruktúry v databáze sa realizuje v obsluhe `onupgradeneeded`
- každý záznam ukladaný do object store (tabuľky) musí mať kľúč

```
request.onupgradeneeded = function(e) {
   let db = e.target.result; // referencia na otvorenu databazu

  // vytvorenie tabulky notes, zaznamy budu jednoznacne
     identifikovatelne autoinkrementujucim id,
  let objectStore = db.createObjectStore('notes',
                        { keyPath: 'id', autoIncrement:true });


  // definovanie indexov, nazov indexu, kluc
  objectStore.createIndex('title', 'title', { unique: false });
};
```

# Indexy

- Index **umožňuje vyhľadávať na hodnoty atribútu** naprieč objektami uloženými v object store
  - teda objekt vieme získať aj cez jeho hodnotu jeho atribútu, nie ibaz cez jeho kľúč
- v spojitosti s **indexami je možné definovať obmedzenie na hodnotu atribútu**, napr. unique príznak
  - máme napr. object store (tabuľku) so zákazníkmi a záznamy (objekty) obsahujú atribút email, vytvorením indexu na daný atribút s príznakom unique zabezpečíme, že žiaden zákazník nebude mať rovnaký email

# Vloženie údajov do DB

- inicializácia transakcie
  - 1. param. – pole object stores, ktorých sa transakcia týka
  - 2. param. – predvolene iba čítanie, ak chceme aj zápis musíme explicitne uviesť `'readwrite'`

```
form.onsubmit = addData;
function addData(e) {
  e.preventDefault();
  let newItem = { title: titleInput.value, body: bodyInput.value };
  let transaction = db.transaction(['notes'], 'readwrite');
  let objectStore = transaction.objectStore('notes');
  var request = objectStore.add(newItem);
  request.onsuccess = function() {};
  transaction.oncomplete = function() {};
  transaction.onerror = function() {};
}
```

# Načítanie údajov cez kľúč objektu

```
var objectStore =
db.transaction('notes').objectStore('notes');
var request = objectStore.get("1");


request.onerror = function(event) {};
request.onsuccess = function(event) {
      console.log(event.target.result.title);
};
```

# Načítanie iterovaním cez cursor

```
let objectStore =
db.transaction('notes').objectStore('notes');


objectStore.openCursor().onsuccess = function(e) {
    // Get a reference to the cursor
    let cursor = e.target.result;
    if(cursor) {
       console.log(cursor.key + " - " + cursor.value.title);
       cursor.continue();
    } else {
       console.log("ziadne dalsie zaznamy")
    }
}
```

# Načítanie cez index

```
var index = objectStore.index("title");
index.get("Kúpiť mlieko").onsuccess = function(event) {
   console.log(event.target.result.body);
};
```

- pozn. ak nie je index unikátny, môže danému kľúču prislúchať viacero záznamov, vtedy vráti prvý záznam, ktorého „hodnota“ je najnižšia

# Vymazanie údajov

```
var request = db.transaction(["notes"], "readwrite")
                .objectStore("notes")
                .delete("1");

request.onsuccess = function(event) {};
```

# Zmena údajov

```
var objectStore =
db.transaction('notes').objectStore('notes');
var request = objectStore.get("1");

request.onerror = function(event) {};
request.onsuccess = function(event) {
   var data = event.target.result;
   data.body = "Iba kravské mlieko";

   var requestUpdate = objectStore.put(data);
   requestUpdate.onerror = function(event) {};
   requestUpdate.onsuccess = function(event) {};
};
```

# Definovanie/štruktúra databázy

- udalosť `onupgradeneeded` je vyvolá inkrementáciou verzie

```
window.indexedDB.open('notes', 2);
```

- v obsluhe `onupgradeneeded` môžeme následne zmeniť schému DB

- **čo ak v jednej karte prehliadača pracujem s verziou 1, a v inej karte sa mi inicializuje verzia 2?**

# Obsluha `onblocked`

```
var openReq = window.indexedDB.open("notes", 2);

openReq.onblocked = function(event){
  alert("Prosím, zatvorte všetky ďalšie
         karty, v ktorých je otvorená táto
         stránka!");
};

openReq.onupgradeneeded = function(event) {
  // ok, vsetky dalsie spojenia na DB su zatvorene,
     mozeme robit zmeny...
};
```

---

```
db.onversionchange = function(event) {
    db.close();
    alert("Nová verzia stránky je pripravená, prosím,
           znovunačítajte stránku!");
};
```

# NPM

- je manažér balíkov/modulov pre JS
  - podobne, ako je composer pre PHP
- je súčasťou Node.js
- inštalácia balíka cez CLI

```
> npm install <package_name>
```

- je možné vytvárať súkromné (private) balíky

# Moduly

- **JS** bol **spočiatku** ako **podporný** skriptovací **jazyk** pri tvorbe webových stránok
  - **dať stránkam viac interaktivity**
- dnes sa webové stránky stávajú webovými aplikáciami
  - tisícky riadkov JS kódu

- moduly nám umožňujú organizovať zdrojový kód
  - **oddeliť funkcionalitu a určiť závislosti**
  - **skryť informácie** a **vystaviť**/exportovať iba verejné **rozhranie**

# Moduly /2

- ďalším z hlavných dôvodov potreby modulov v JS je globálny menný priestor
  - ktorý sa môže ľahko „znečistiť“
- (*Vanilla*) ES5 nemá podporu modulov
  - **potrebujeme tzv. formát a závadzač modulu (modules formats and loaders)**
  - sú to knižnice 3. strán, ktoré nám umožňujú organizovať JS kód do modulov

# Formát a zavádzač modulu

- formát modulu špecifikuje jeho konkrétnu syntax
  - tá je spravidla odlišná od natívneho JavaScriptu,
  - preto je potrebný zavádzač na interpretovanie danej syntaxe modulu
  - AMD, CommonJS formáty
- zavádzač modulu je nástroj /knižnica/ 3. strany
  - dokáže interpretovať formát modulu
  - RequireJS, SystemJS zavádzače

- ES6 má natívnu podporu modulov
  - formát modulu je natívny a teda nie je potrebný zavádzač
- porovnanie formátov

# Formát AMD

- Asynchronous Module Definition
- spravidla používaný na strane klienta
  - skripty, ktoré interpretuje prehliadač (oproti JavaScriptu vykonávanom na serveri)
- AMD formát definuje novú funkciu `define`, má 2 parametre
  - pole závislostí
  - definíciu funkcie
- pozn. `define` nie je súčasťou JS, je potrebný zavádzač

# AMD - súbor playboard.js

```
define([], function() {
  console.log('Vytvorenie nového modulu playboard');
  function showState() { ... }
  function update() { ... }
  // spristupnujeme/exportujeme showState a update
  return {
      showState: showState,
      update: update
  }
});
```

# AMD – súbor `game.js`

```
// 1. parameter pole závislosti
// nie je potrebná extenzia .js
// 2. parameter definícia funkcie
define(['./player', './playboard'],
      function(player, playboard) {
    …
})
```

# Zavádzač RequireJS

- do HTML vložíme odkaz na RequireJS závádzač

```
<script
      data-main="js/app"
      src="node_modules/requirejs/require.js">
</script>
```

- `data-main` **špecifikuje prístupový bod** pre danú aplikáciu (tzv. entry point)

- `npm install --save bower-requirejs`

# Formát CommonJS

- spravidla používaný na strane servera
  - NodeJS aplikácie

- definuje objekt `module.exports`,
  - pomocou ktorého určíme verejné rozhranie (sprístupníme informácie modulu)

# CommonJS – playboard.js

```
console.log('Vytvorenie nového modulu playboard');
function showState() { ... }
function update() { ... }

// spristupnujeme/exportujeme showState a update
module.exports = {
    showState: showState,
    update: update
};
```

# CommonJS – game.js

```
var playboard = require('./playboard.js');

playboard.showState();
```

# Zavádzač SystemJS

- do HTML vložíme odkaz na SystemJS závádzač

```
<script
   src="node_modules/systemjs/dist/system.js">
</script>
```

- nastavíme základnú konfiguráciu
  (SystemJS má podporu viacerých formátov)

```
<script>
  System.config({
    meta: {
      format: 'cjs'  // CommonJS format
    }
  });
  System.import('js/app.js'); // root module
</script>
```

- `npm install --save systemjs`

# Moduly v ES6 - `export`

- 2 možné spôsoby

```
export function setName(name) { ... }
export function getName() { ... }

// alebo
function setName(name) { ... }
function getName() { ... }
export {setName, getName};
```

# Moduly v ES6 - `import`

- 2 možné spôsoby

```
// importujeme cely modul
import * as playboard from './playboard.js';
// playboard.update();

// alebo iba podmnozinu z moznych
// exportovanych elementov,
// urcime konkretne elementy
import {
      getName as getPlayerName,
      logPlayer
} from './player.js’;
// getName();
```

# Transpilátor - transpiler

- v ES6 máme moduly natívne
  - pre moderné prehliadače (najnovšie verzie) nepotrebujeme riešenia tretích strán
- JS sa vyvíja, nová špecifikácia/nové vlastnosti každý rok
- implementácia špecifikácie je ale pozadu
- chceme ale programovať v najnovšej verzii jazyka
  - využívať nové vlastnosti
- môžeme, ale potrebujeme pretransformovať kód
  - napísaný v najnovšej špecifikácii JS do ekvivalentného kódu, ktorú poznajú súčasné interpretery
  - potrebujeme transpilátor

# Transpilátor /2

- transpilátor preloží napr. ES6 kód do ES5 kódu
- jedným z najpopulárnejších transpilátorov je babel
- umožňuje nastaviť preset, teda verziu ES, z ktorej má spraviť transpiláciu
  - es2015, es2016, es2017, latest
- pozrite si ukážku prekladu z ES6 do ES5
- na stránke nájdete aj sumarízáciu a príklady nových vlastností ES6
- `npm install --save-dev babel-cli`
- `npm install --save-dev babel-preset-es2015`

# Balíkovač - bundler

- zabalí niekoľko zdrojov do jedného – bundle.js
- napr.  z niekoľkých JS súborov/modulov vytvorí jeden súbor
- jedným z najpopulárnejších balíkovačov je browserify

- na jednej strane sa môže znížiť réžia s postupným sťahovaním viacerých súborov
- na strane druhej, pozor na zaobalenie veľkého množstva súborov
  - môže sa tým významne zvýšiť čakanie používateľa na moment, kedy už niečo konečne na stránke uvidí

- `npm install --save-dev browserify`

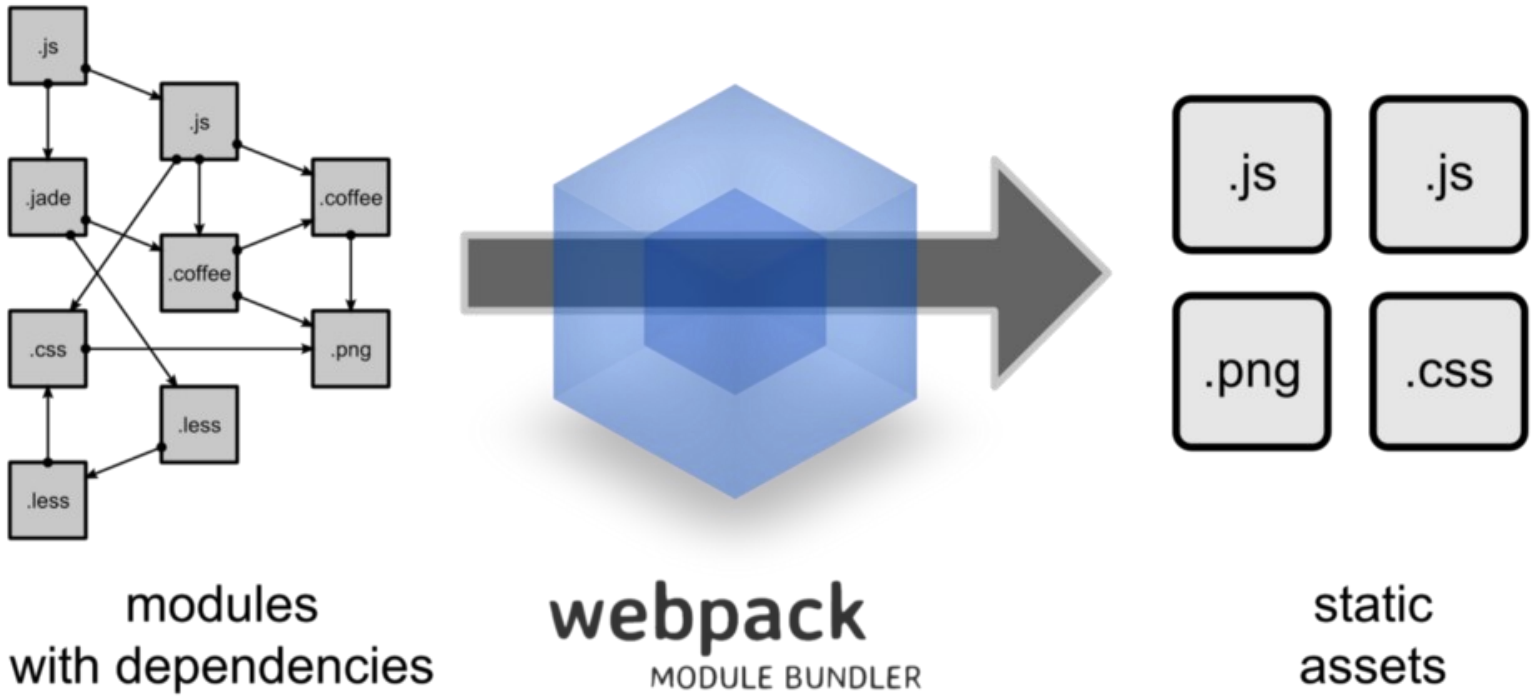
# Webpack

- potrebujeme kód transpilovať
- chceme vytvoriť balík
- chceme kód minifikovať
- chceme kód zamlžiť (obfuscation) / deobfuscation

- Webpack umožňuje všetko v jednom kroku,
  a ešte ďaleko viac...

- `npm install --save-dev webpack`

# Webpack


.js
.js
.jade
.coffee
.coffee
.css
.png
.less
.less
modules
with dependencies
webpack
MODULE BUNDLER
.js
.js
.png
.css
static
assets

# Webpack - config

- prístupový bod
- výstupný súbor a umiestnenie
- regexp výrazy na výber súborov, ktoré má vykonať zavádzač
- súbory (napr. celý priečinok), ktoré nemá zavádzač vykonať (napriek tomu, že spĺňajú regexp)
- aký typ zavádzača sa má použiť
- predvoľby/nastavenia, s ktorými má byť zavádzač vykonaný

# Webpack – config príklad

```
module.exports = {
    entry: './js/app.js', // entry point aplikacie
    output: {
       path: './build', // umiestnenie/priecinok pre vystupny subor
       filename: 'bundle.js', // nazov vystupneho suboru
    },
    module: {
      loaders: [{
         test: /\.js$/, // js subory
         exclude: /node_modules/, // vyluc tie v danom priecinku
         loader:'babel-loader', // pouzi zavadzac babel
         query: {
           presets: ['es2015'] // pouzi dane nastav. v zavadzaci
         }
      }]
    }
};
```

# Webpack – plugin - minifikácia

- minifikácia - UglifyJS Webpack Plugin

```
const UglifyJsPlugin =
            require('uglifyjs-webpack-plugin‘)

module.exports = {
  plugins: [
    new UglifyJsPlugin()
  ]
}
```

# Webpack – plugin - zamlženie

- zamlženie - javascript-obfuscator webpack

```
const JavaScriptObfuscator = require('webpack-obfuscator');
module.exports = {
    entry: {
        'abc': './test/input/index.js',
        'cde': './test/input/index1.js'
    },
    output: {
        path: 'dist',
        filename: '[name].js' // output: abc.js, cde.js
    },
    plugins: [
        new JavaScriptObfuscator({
            rotateUnicodeArray: true
        }, ['abc.js‘]) // excluded
    ]
};
```

# JavaScript Strict mode - od ES5

- prepnutie prehliadača do striktného režimu interpretovania JS
- zbavuje JS niektorých zákerných nástrah, ktoré nevyvolajú chybu (výnimku)
- zakazuje používať syntax, ktorá bude možno použitá v budúcich verziách ECMAScriptu
  - predpríprava na budúci vývoj ES
- striktný režim je možné aplikovať na celý skript, alebo na funkciu

# Strict mode – hlásenie chýb

- znemožňuje omylom vytvoriť globálnu premennú, preklep v názve premennej v normálnom JS vytvorí na globálnej úrovni novú premennú

```
mistypedVarai ble = 17; // ReferenceError
```

- priradenie, ktoré by nefungovalo, ale prešlo vyhodí výnimku

```
NaN = 42;                          // TypeError
var obj = { get x() { return 17; } };
obj.x = 5;                         // TypeError
var fixed = {};
Object.preventExtensions(fixed);
fixed.newProp = "ohai";            // TypeError
```

# Strict mode – hlásenie chýb /2

- vymazanie atribútu, ktorý nie je možné vymazať

```
delete Object.prototype; // TypeError
```

- všetky atribúty objektu musia mať unikátny názov
  - v normálnom JS určuje hodnotu poslednú z nich

```
var o = { p: 1, p: 2 }; // syntax error
```

- argumenty funkcie majú rôzne identifikátory
  - v normálnom JS skryje posledný argument všetky predošlé s rovnakým názvom (`arguments[i]`)

```
function sum(a, a, c) // syntax error
{
  "use strict";
  return a + b + c; // chyba, keby mal bol tento kod vykonany
}
```

# Strict mode – hlásenie chýb /3

- zakazuje zápis čísel v osmičkovej sústave

```
var sum = 015 + // syntax error
          197 +
          142;
```

- pozn., zápis čísel v osmičkovej síce nie je súčasťou ES, ale je podporovaný v prehliadačoch
  - začínajúci vývojári sa mylne domnievajú, že počiatočná nula nemá žiadny význam, takže nuly používajú k „zarovnaniu“, čo ale zmení hodnotu čísla

# Strict mode – celý skript

- na začiatku skriptu uvedieme

```
"use strict";
```

- pozor, pripojenie skriptu so `strict` ku skriptu bez `strict` znamená, že výsledný skript je v režime `strict`
  - ak non strict + strict -> non strict

# Strict mode – vo funkcii

```
function myFunc()
{
  // strict režim plati iba vo funkcii
  'use strict';

  // ale plati aj vo vnorenej funkcii
  function nested() {}
}
```

# Strict mode – ďalšie ...

- **Strict mode - ďalšie obmedzenia...**

- režim `strict` predstavuje obmedzenia, ktoré uľahčia vývoj nových špecifikácii ES, a teda JS ako takého
- podpora prehliadačov

# Vue.js

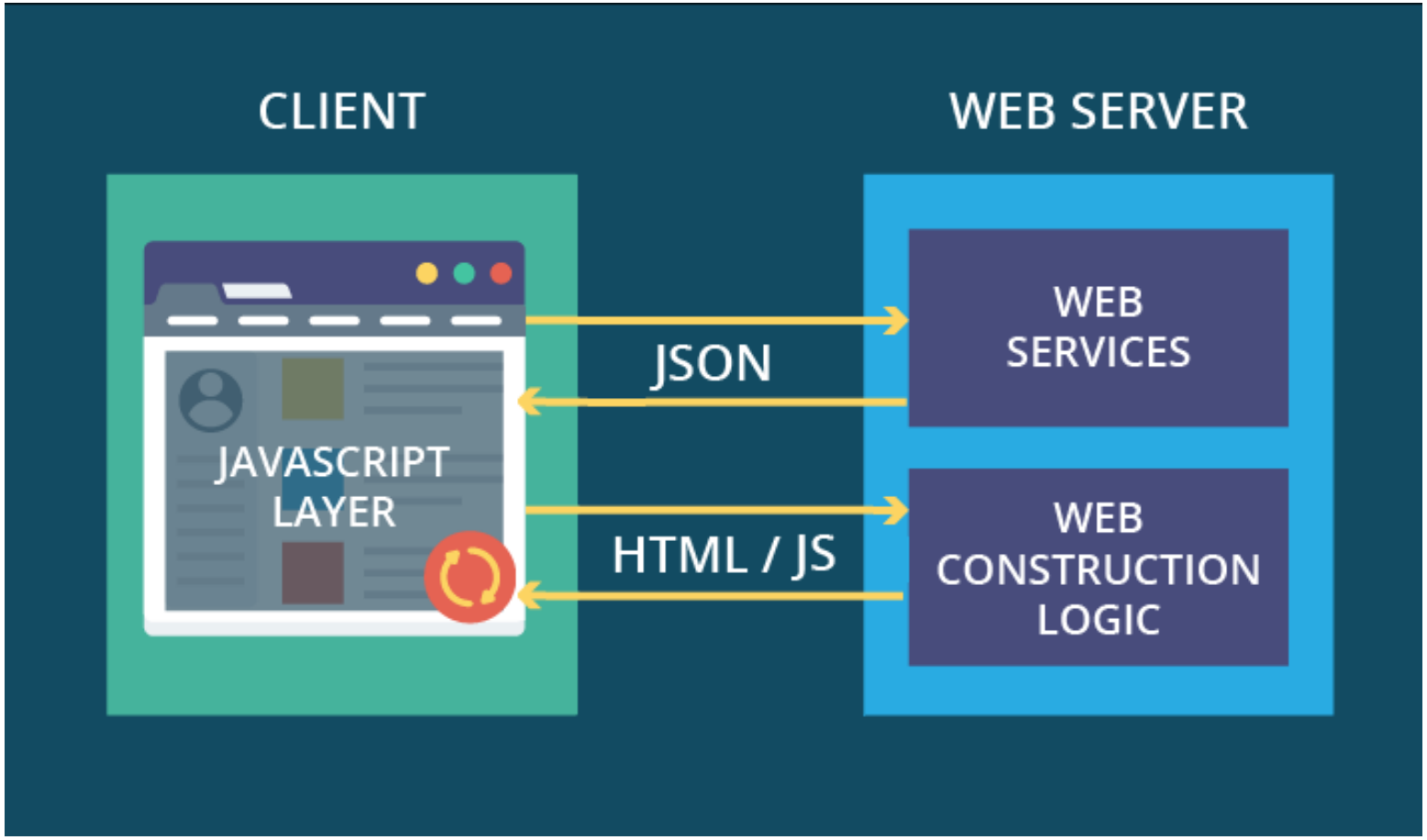
- populárny progresívny JS rámec na tvorbu plne-interaktívneho používateľského rozhrania webových stránok/aplikácií
- vytvoril ho Evan You po tom, ako pracoval v Googli, kde používal AngularJS
  - extrahoval a preniesol do Vue najlepšie koncepty z Angularu
  - prvá oficiálna verzia 2014

porovnanie s inými rámcami

# Client-side rendering

- **server vygeneruje HTML stránku, ktorá obsahuje koreňový element (kontajner) pre JavaScript aplikáciu/vrstvu**
- spravidla celá biznis logika - konštrukcia/generovanie stránok (rozhrania) je na klientovi – tučný klient
  - napísaná v JS, použitím rámca Angular, React, Vue...
  - JavaScript generuje HTML, aplikuje štýly, defiunuje správanie
- server vystavuje webové služby, ktoré poskytujú iba údaje (napr. JSON), ktorými sa napĺňa rozhranie aplikácie

# Client-side rendering
# Single Page Application - SPA



# Client side

- po úvodnom načítaní je množstvo prenášaných údajov minimálne, rozhranie kompletne generuje JavaScript
- **na vývoj je potrebná výborná znalosť JavaScriptu a špecializovaného rámca (Vue)**
- mínus: bezpečnosť – celá logika je na klientovi, poskytujeme kompletný kód

# model–view–**viewmodel**

```
<body>
    <div id="app">
       {{message}}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue"></script>

    <script>
      var vm = new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue!'
        },
      });
    </script>
</body>
```

**viewmodel**

- inštancia vue, objekt, ktorý synchronizuje rozhranie a model (dáta)

# model–view–viewmodel

```
<body>
    <div id="app">
       {{message}}
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue"></script>

    <script>
      var vm = new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue!'
        },
      });
    </script>
</body>
```

**model**

- (dátový) objekt jazyka JavaScript
- `vm.$data`

# model–**view**–viewmodel

```
<body>
    <div id="app">
       {{message}}
    </div>

    <script src="https://cdn.jsdeli

    <script>
      var vm = new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue!'
        },
      });
    </script>
</body>
```

**view (rozhranie)**

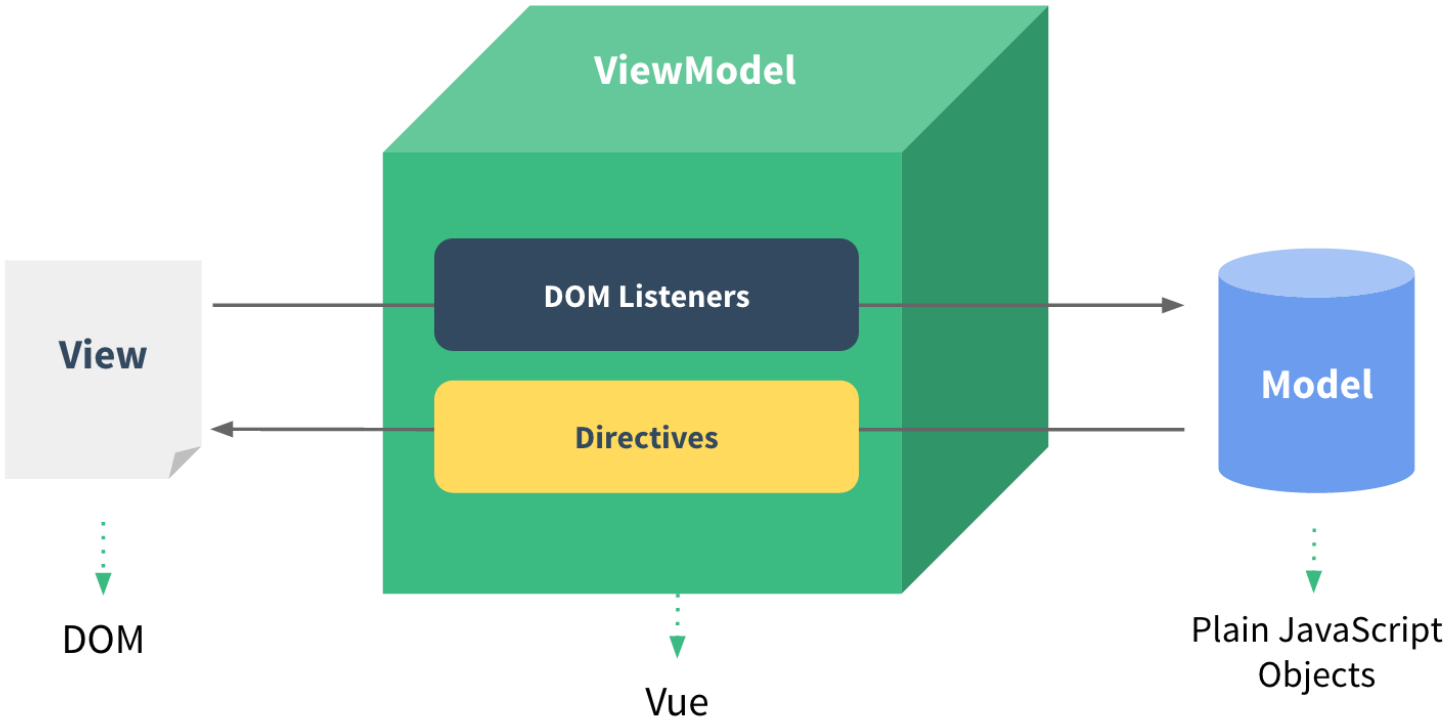
- DOM element, každá inštancia vue je prepojená so zodpovedajúcim DOM elementom
- `vm.$el`
- pri vytvorení inštancie je všetkým potomkom daného elementu inicializované reaktívne dátové previazanie (data binding)

# Reaktivita

- zriedkavo je potrebné sa dotknúť DOMu ako takého
  - okrem vlastných direktív (custom directives)

- **zmena údajov (dátových objektov) automaticky vyvolá aktualizáciu (update) rozhrania**
  - reaktivitu je možné naviazať až na úroveň textových uzlov (textNode)
  - aktualizácia rozhrania je vykonávaná asynchrónne pre vyšší výkon

# Model–view–viewmodel



# Komponenty

- komponent je základnou stavebnou jednotkou
- je určený
  - **šablónou** (html + css) - ako vyzerá, napr. tlačídlo
  - **a správaním** – čo sa má vykonať,
    napr. keď používateľ klikne na tlačidlo

# Komponent - šablóna

```
<div id="vue-app">
      <h1>{{ heading }}</h1>
      <p>
            {{ message }}
      </p>
      <button>Klikni na mňa</button>
      Klikli ste {{ clickCounter }}-krát.
</div>
```

# Komponent - správanie

- správanie je definované JavaScriptom

```
var app = new Vue({
  el: '#vue-app',
  data: {
    heading: 'Počítadlo kliknutí',
    message: 'Tento komponent zobrazuje
          počet kliknutí na tlačidlo',
    clickCounter: 0
  }
});
```

# Komponent – načúvanie na udalosť

- je potrebné zaistiť, aby sa po kliknutí na tlačidlo vykonala nejaká akcia
  - inkrementácia počítadla o 1

```
<button v-on:click="buttonClick()">
    Klikni na mňa
</button>
```

# Komponent – dodefinovanie akcie

- správanie komponentu je potrebné rozšíriť o akciu `buttonClick()`

```
methods: {
    buttonClick: function() {
        this.clickCounter++;
    }
}
```

# Komponent – reaktívne správanie

- keď budeme klikať na tlačidlo, inkrementuje sa premenná **`clickCounter`** v dátovom objekte
- **táto zmena automaticky vyvolá aktualizáciu (update) rozhrania**

```
var app = new Vue({
  el: '#vue-app',
  data: {
    heading: 'Počítadlo kliknutí',
    message: 'Tento komponent zobrazuje
              počet kliknutí na tlačidlo',
    clickCounter: 0
  }
});
```

# Direktívy

```
<button v-on:click="buttonClick()">
    Klikni na mňa
</button>
```

- podmienené zobrazenie nejakého obsahu

```
<button v-if="clickCounter < 5"
        v-on:click="buttonClick()">
    Klikni na mňa
</button>
```

# Direktíva – v-else

```
<div v-if="clickCounter < 5" >
    <button v-on:click="buttonClick()">
      Klikni na mňa
    </button>
    Klikol si {{ clickCounter }}-krát.
</div>
<div v-else>
    Sorry, už by stačilo klikania...
</div>
```

# Direktíva – v-else-if

```
<div v-if="clickCounter < 5" >
    <button v-on:click="buttonClick()">
       Klikni na mňa
    </button>
    Klikol si {{ clickCounter }}-krát.
</div>
<div v-else-if="clickCounter < 15" >
    <button v-on:click="buttonClick()">
       Klikaj ďalej, zotrvaj!
    </button>
    Klikol si {{ clickCounter }}-krát.
</div>
<div v-else>
     Sorry, už by stačilo klikania...
</div>
```

# Direktíva – `v-show`

- `v-if` – pokiaľ je podmienka `false`, element je odstránený z DOMu stránky
- `v-show` – element nie je odstránený z DOMu stránky, ale má nastavený CSS `display: none;`

- môže byť výhodné napr., keď máme zložitý element (čo do štruktúry , počtu potomkov) a často sa mení podmienka, vtedy musí prehliadač pri každej zmene element odstrániť, resp. pridať
- ale, element môže spôsobiť vyššiu réžiu pri prvotnom vykreslení stránky, pretože sa vykresľuje vždy, a príp. sa skryje, v prípade `v-if` ho bude generovať iba ak sa má skutočne zobraziť

# Direktíva – `v-for`

- iterovanie zoznamom/polom
- pridajme do komponentu v dátovom objekte pole `items: [1,3,5,7,9]`
- rozšírme šablónu

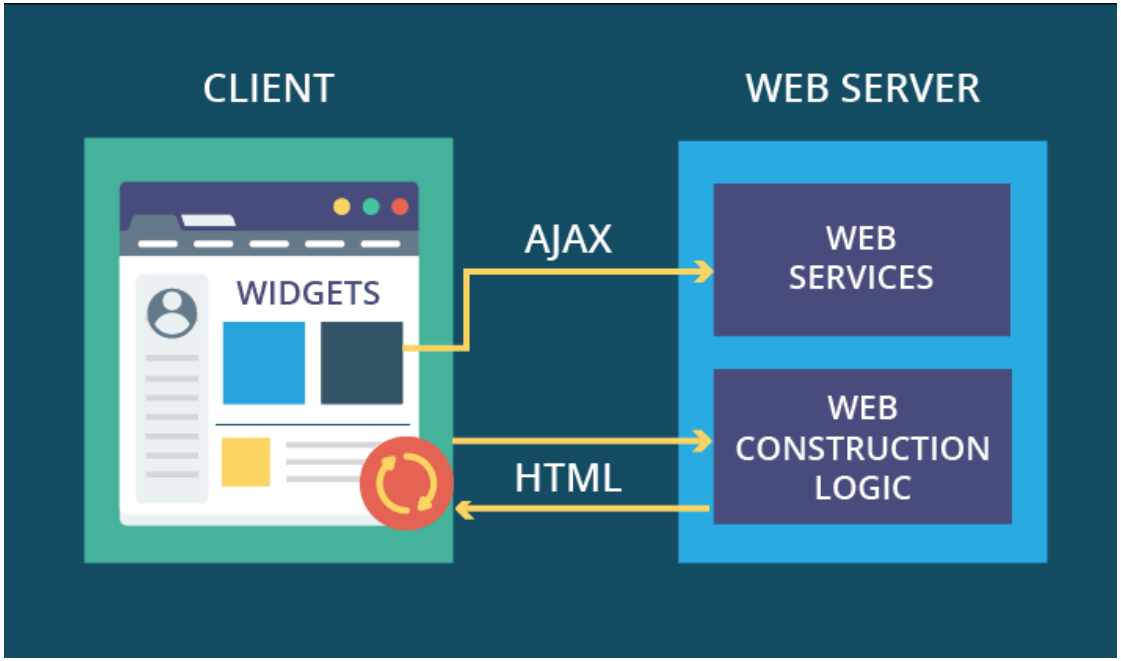
```
<ul>
      <li v-for="item in items">
        {{ item }}
      </li>
</ul>
```

# Direktíva – v-for /2

```
<ul>
      <li v-for="(item, index) in items"
          v-on:click="removeClick(index)" >
        {{ item }}
      </li>
</ul>
```

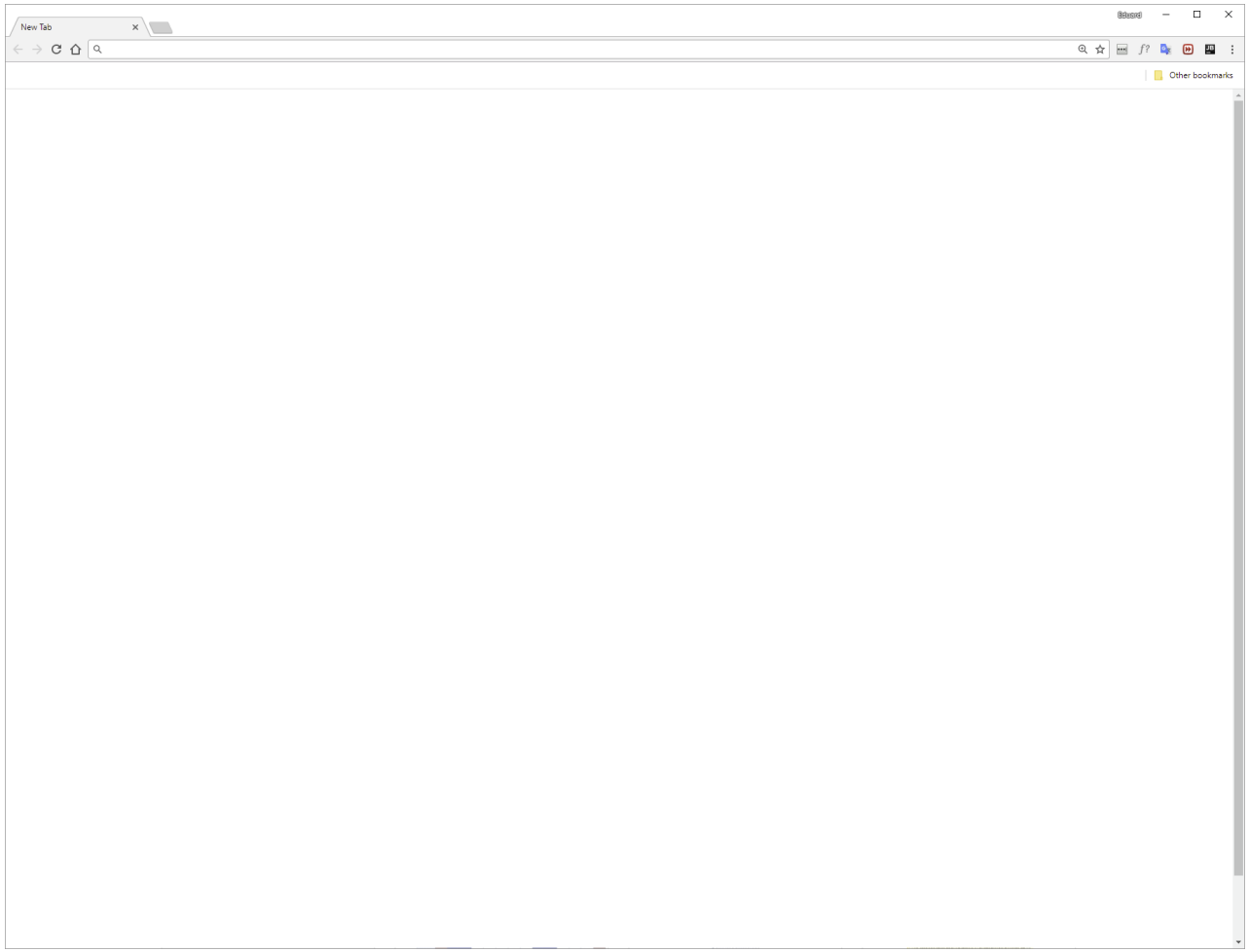
# JS widgets /komponenty

- nemusíme mať „čistú“ client-side architektúru
- Vue môžeme použiť aj v JS widgets architektúre
  - kde widgety sú komponenty



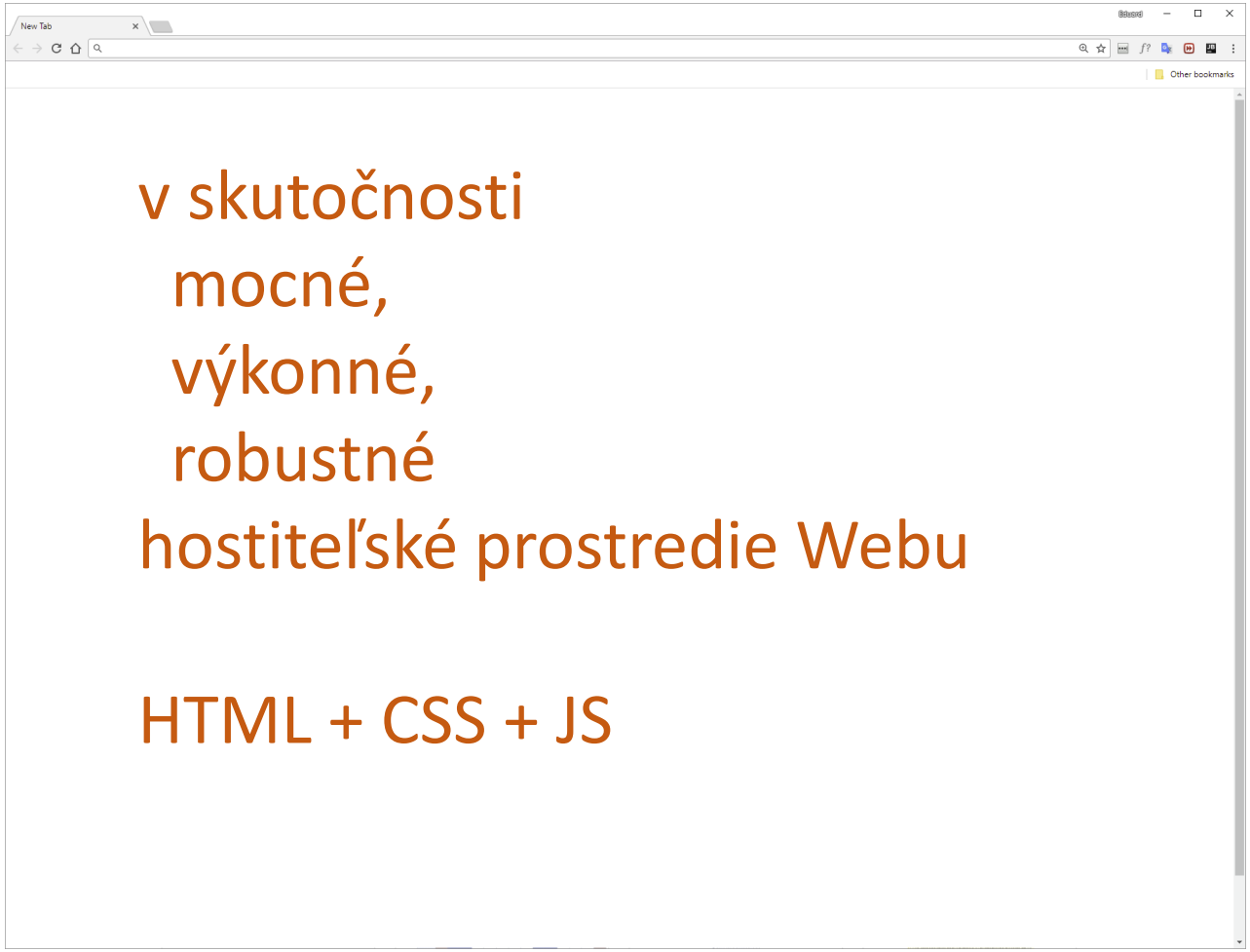
# Prehliadač

- na prvý pohľad takmer nič



# Prehliadač

- na prvý pohľad takmer nič



# Zdroje

- Eloquent JavaScript
- ECMAScript 2015 features
- Learning JavaScript Design Patterns

- Functional Programming in JavaScript