



JavaScript Web API, CSR

Základy webových
technologií

Eduard Kuric

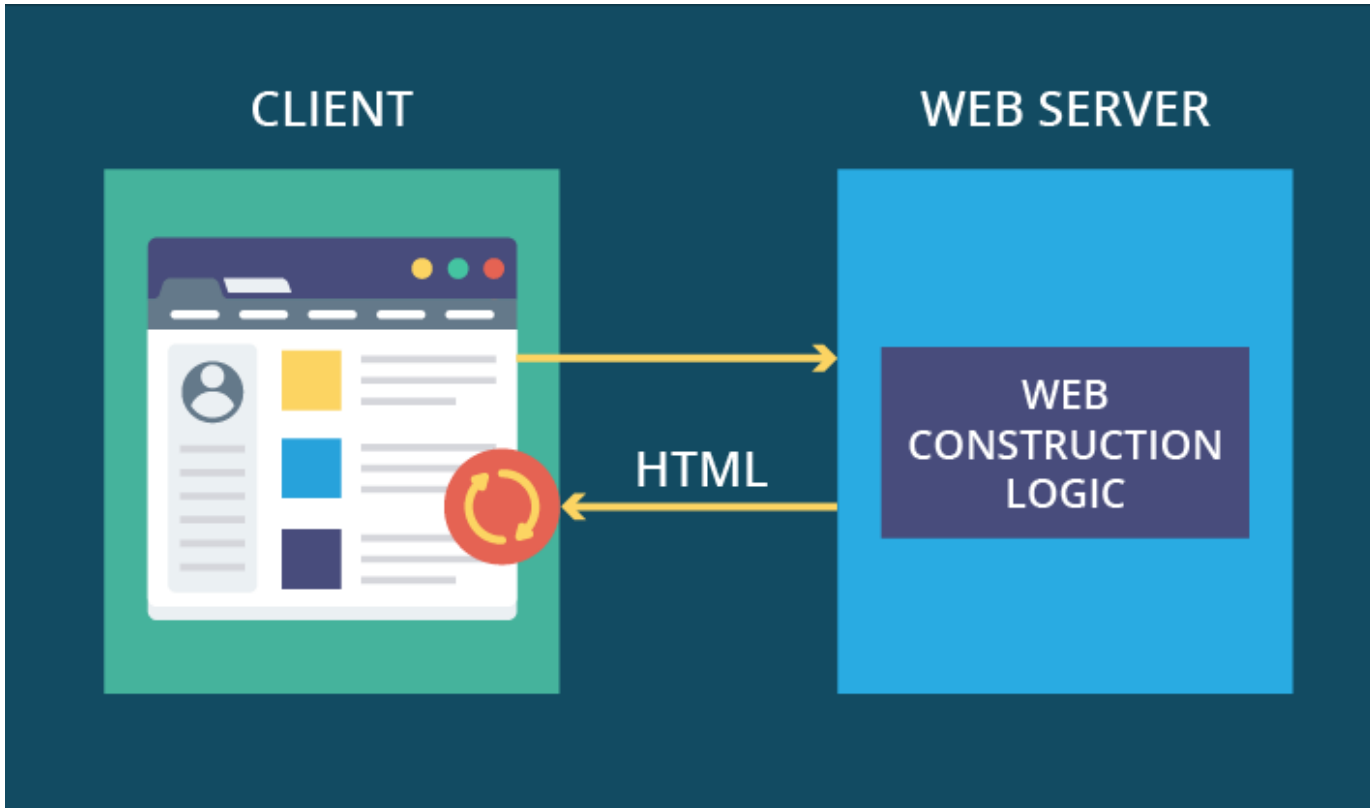
- Web API
 - Získavanie údajov zo servera (AJAX, Fetch API)
 - JSON, Promises
 - Ukladanie údajov na strane klienta (sessionStorage, localStorage, IndexedDB)
- Moduly
 - AMD (RequireJS), CommonJS (SystemJS), ES6
- Transpilátor (transpiler - Babel)
- CSR

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)

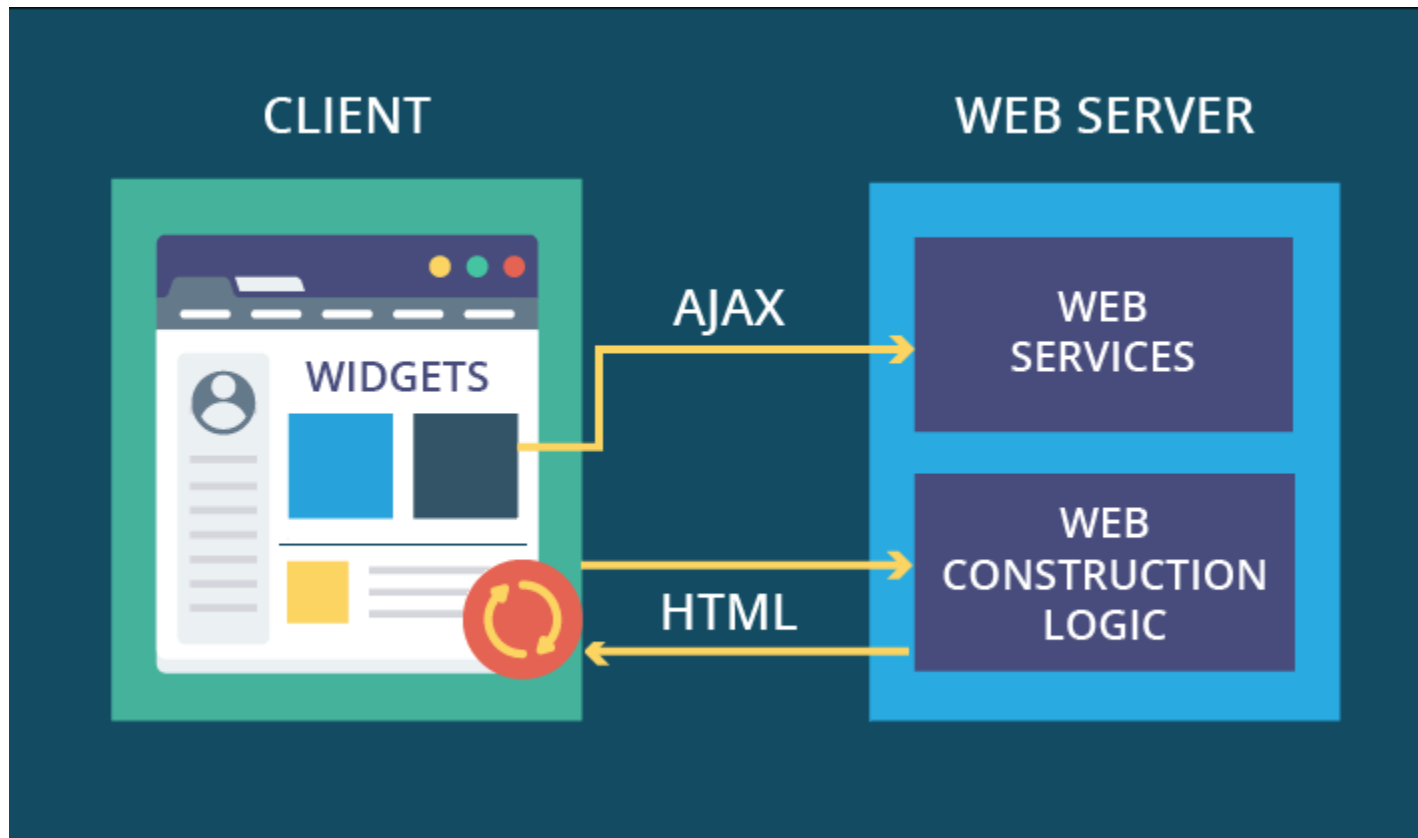
SSR – Server-side rendering



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



JSON – formát údajov

- JavaScript Object Notation
- textový formát údajov
- vhodný na zápis (krátkych) štruktúrovaných údajov **vymieňaných 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 reálne, 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
 - JS 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 prísľub 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`

Prísľuby sú asynchrónne

```
var askMom = function () {  
    console.log('Mama, kúpiš mi vláčik?');  
  
    willIGetNewTrain  
        .then(showOff)  
        .then(function (fulfilled) {  
            console.log('Kúpim ti nový vláčik.'));  
        })  
        .catch(function (error) {  
            console.log('Neposlúchaš, nedostaneš vláčik.');        });  
  
    console.log('Chcem radšej nové auto.');};  
Askmom()
```

Sľuby sú asynchrónne /2

- očakávali by sme
 - **Mama, kúpiš mi vláčik?**
 - Kúpim ti nový vláčik.
 - **Chcem radšej nové auto.**
- v skutočnosti môže byť aktuálne poradie:
 - **Mama, kúpiš mi vláčik?**
 - **Chcem radšej nové auto.**
 - Kúpim ti nový vláčik.

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 požiadava 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 obsluhu `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 tabuľky 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 iba 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 obsluhu `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, všetky ďalšie spojenia na DB sú zatvorene,  
    // môžeme robiť 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 funkcionality 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“
- ES5 nemala 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/vanilla 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
- pravidla 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() { ... }  
    // sprístupnujeme/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 zavádzač](#)

```
<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

- pravidla 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() { ... }  
  
// sprístupnujeme/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 zavádzač](#)

```
<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 celý modul
import * as playboard from './playboard.js';
// playboard.update();
```

```
// alebo iba podmnožinu z možných
// exportovaných elementov,
// určíme konkrétne 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

- umožňuje používať najnovšiu verziu JavaScriptu v súčasnosti
- transpilátor preloží najnovší kód do kompatibilného kódu
- jedným z najpopulárnejších transpilátorov je [babel](#)



Vue.js

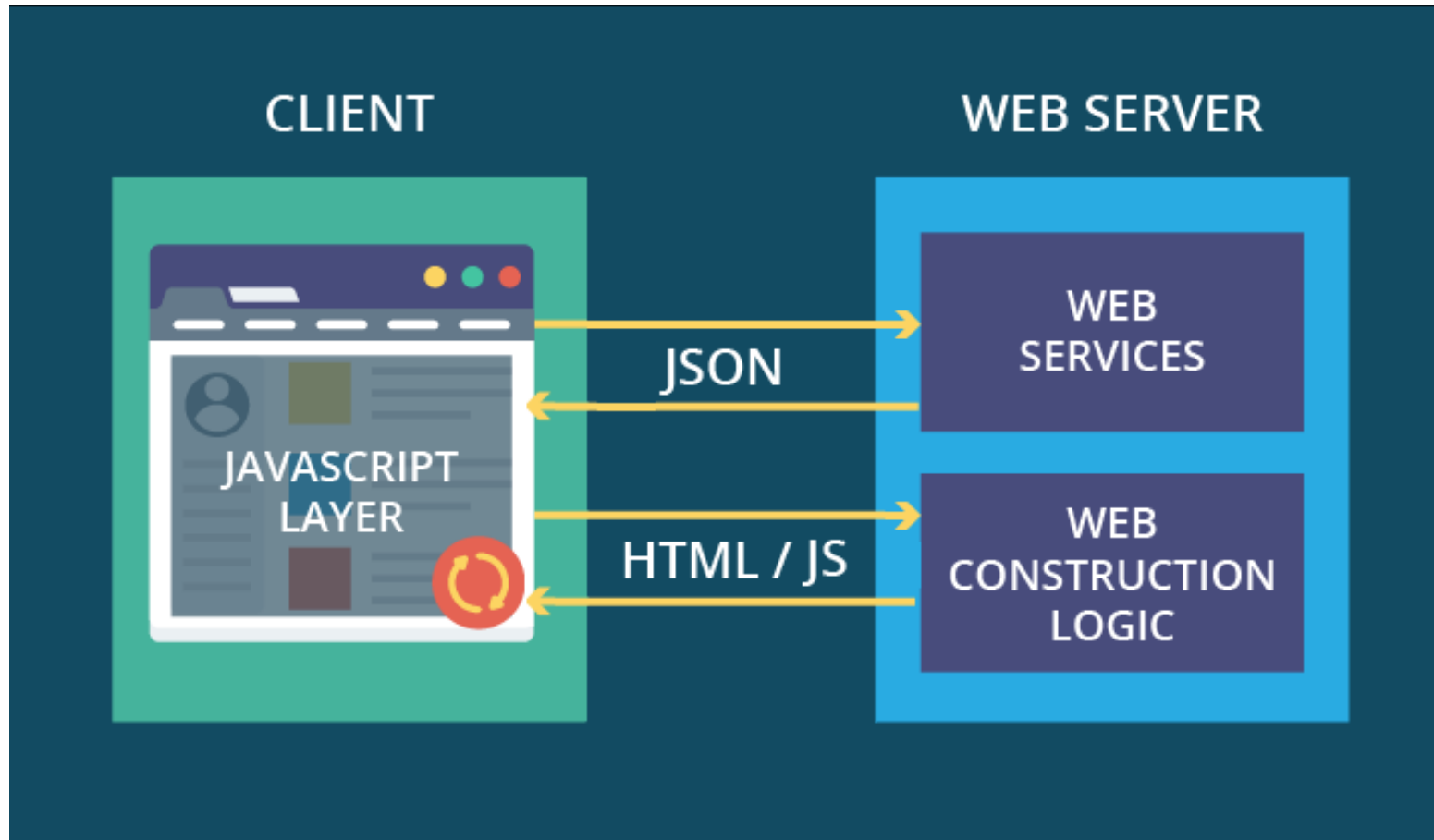
- populárny progresívny JS rámec na tvorbu plne-interaktívneho používateľského rozhrania webových stránok/aplikácií
- vytvoril ho Evan You po tom, ako pracoval v Googli, kde používal AngularJS
 - extrahoval a preniesol do Vue najlepšie koncepty z Angularu
 - prvá oficiálna verzia 2014

[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, defiňuje správanie
- **server vystavuje webové služby, ktoré poskytujú iba údaje (napr. JSON), ktorými sa naplňa rozhranie aplikácie**

Client-side rendering Single Page Application - SPA

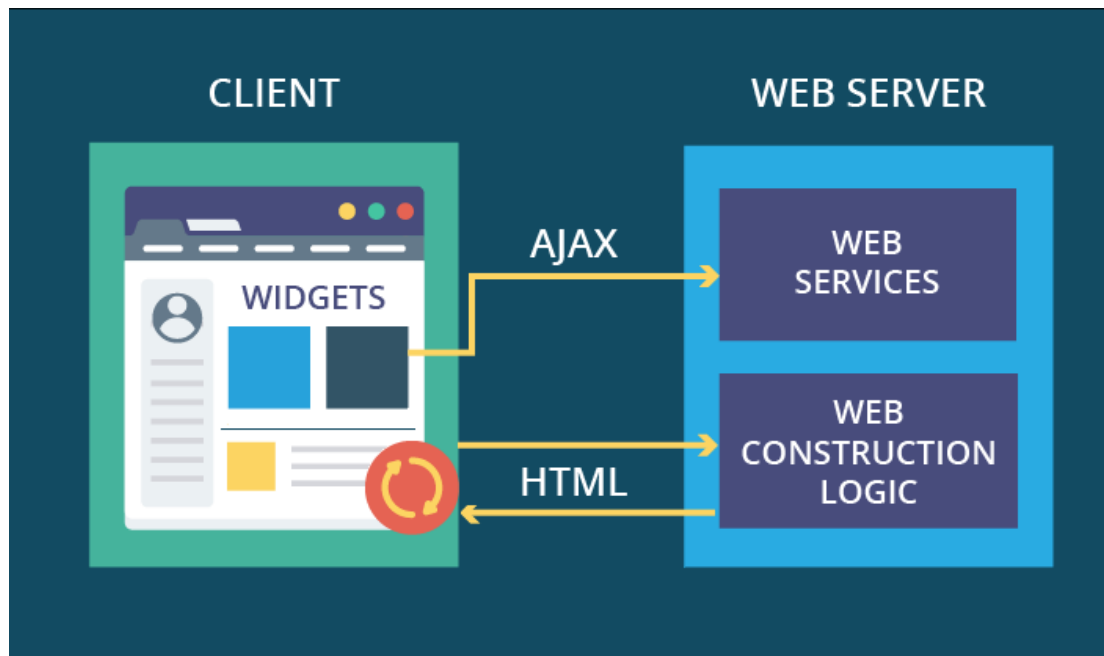


Client side

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

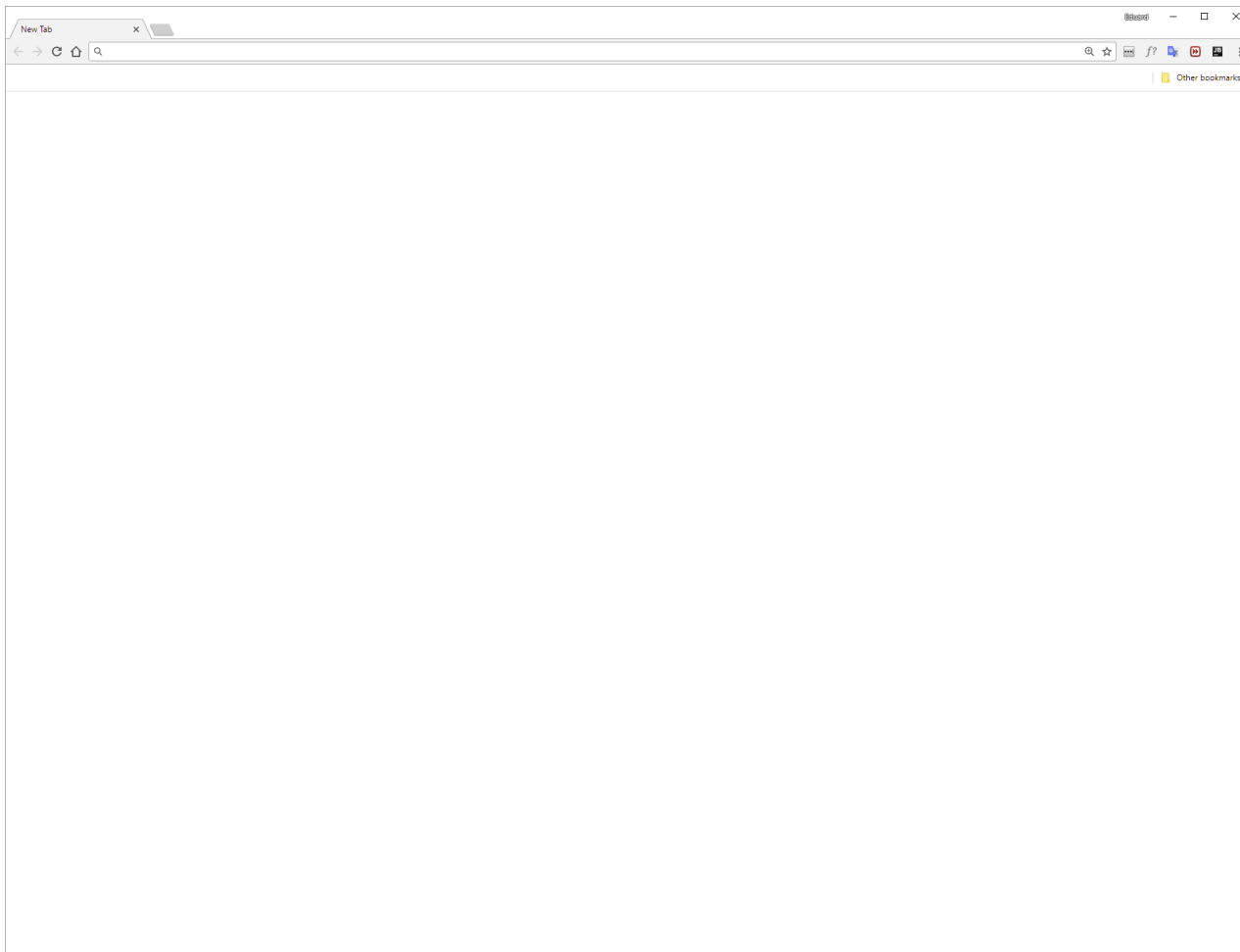
JS widgets /komponenty

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



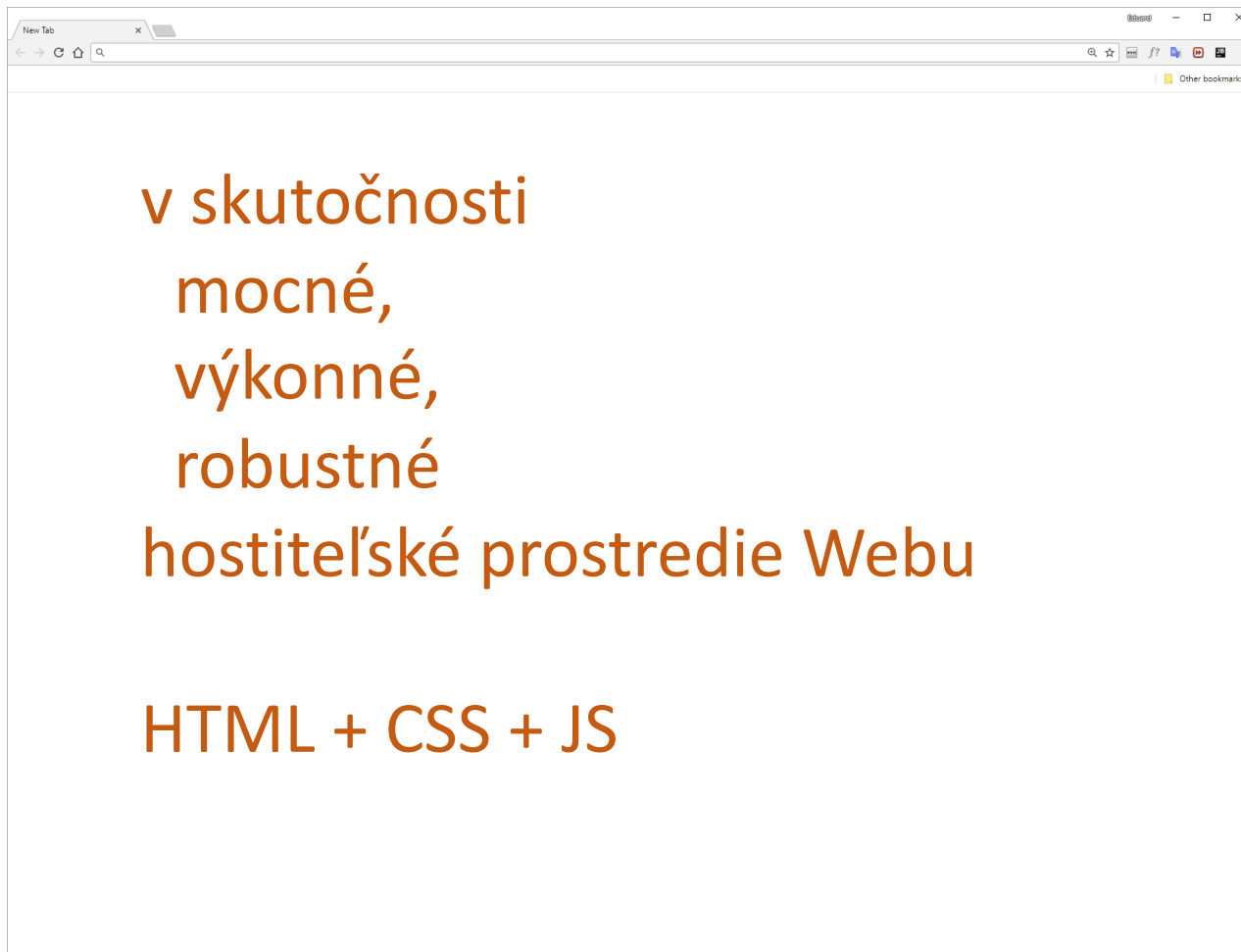
Prehliadač

- 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](#)