Webové technológie 8

Aplikácie Webových technológií

Peter Bednár

Uchovanie dát na strane klienta

- Objekty `window.localStorage` a `window.sessionStorage`
- Umožňujú ukladať dáta v prehliadači medzi kliknutiami na rôzne stránky (z tej istej domény)
  - Dvojice kľúč:hodnota, hodnoty sú reťazce
  - Každá doména má vlastné hodnoty
  - Používajú sa pre uloženie preferencií a nastavení používateľov – nie citlivé autentikačné údaje

Objekty `window.localStorage` a `window.sessionStorage`

- `window.localStorage`
  - Údaje nemajú expiráciu – ostávajú v prehliadači aj po zavretí okna kých ich aplikácia alebo používateľ nevymaže
- `window.sessionStorage`
  - Automaticky sa vymažú po skončení sedenia (*session*)
  - Sedenia sa týka iba jednej karty prehliadača

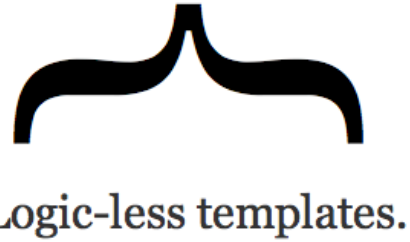
- K dátam sa pristupuje ako k mape, napr.:

```
window.localStorage["preferredLanguage"] = "sk";
window.localStorage.removeItem("preferredLanguage");
```

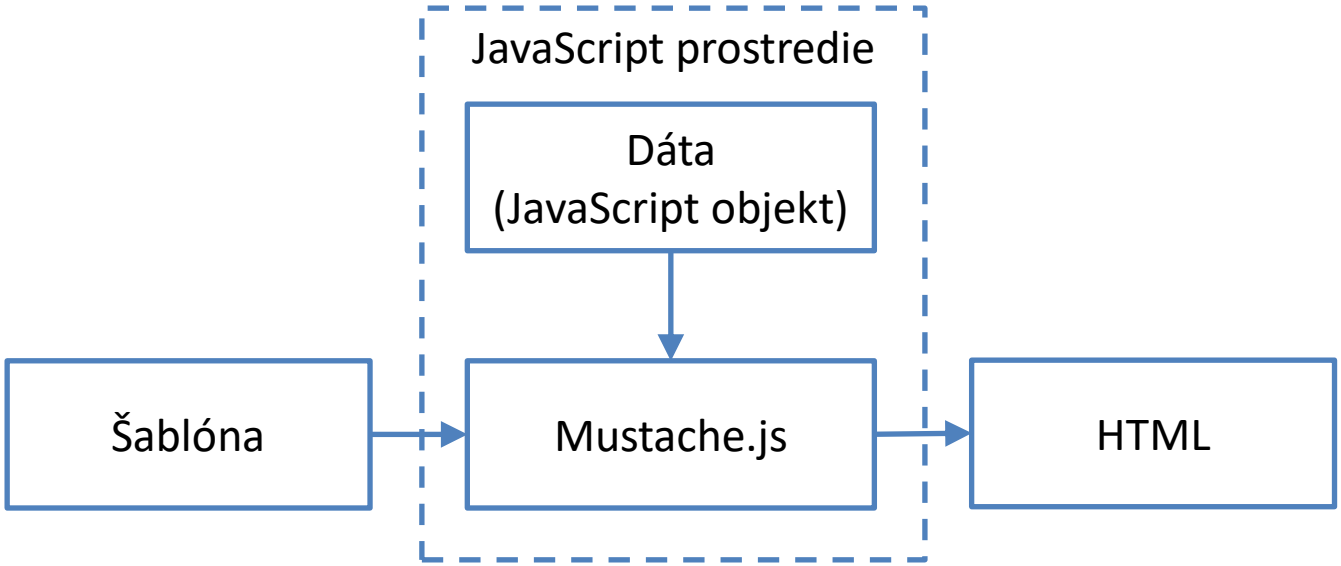
Generovania obsahu pomocou šablón

- Pomocou šablón je možné ľahko definovať v HTML kóde premenné, ktoré sa dynamicky doplnia podľa aktuálnych dát
  - Zjednodušuje sa návrh a implementácia dynamických stránok

- Mustache
  - Jednoduchý šablonovací systém
  - Implementovaný pre rôzne jazyky
  - Generovanie stránok na strane servera aj klienta
  - Stránka projektu: https://mustache.github.io/
  - JavaScript implementácia: https://github.com/janl/mustache.js/



Generovania obsahu pomocou šablón na strane klienta



Mustache + JavaScript Príklad (1)

1. Definovanie šablóny v HTML:

```
<script id="name-templ" type="text/template">
  <p>
    {{name}} {{surname}}
  </p>
</script>
```

2. Načítanie Mustache JavaScript knižnice:

```
<script src="js/mustache.min.js"/>
```

Mustache + JavaScript Príklad (2)

3. Generovanie HTML zo šablóny pre dané dáta:

```
// dáta
var data = { name: "Peter", surname: "Bednár" };
// načítame šablónu do reťazca templ
var templ = document.getElementById("name-templ").
    innerHTML;
// vygenerujeme HTML zo šablóny
var html = Mustache.render(templ, data);
// nastavíme obsah stránky
document.body.innerHTML = html;
```

Mustache šablóny (1)

- {{vlasnosť}} – zobrazenie hodnoty dátovej vlastnosti
- {{vlasnosť.vlastnosť}} – vnorené objekty
- {{^vlasnosť}} – vynechané hodnoty
- Príklad

```
var data = { address: {
  street:"Letná", number:"9", ZIP:"04200", city:"Košice" }
};

{{#address}}
  Adresa: <br>
  {{street}} {{number}}, {{ZIP}} {{city}}
{{/address}}
{{^address}}
  (neznáma)
{{/address}}
```

Mustache šablóny (2)

- {{#vlasnosť}} ... {{/vlasnosť}} – zobrazenie zoznamu objektov
- Príklad:

```
var data = { employees: [
    { firstName: "Peter", lastName: "Bednár" },
    { firstName: "John", lastName: "Smith" } ]
};

<ul>{{#employees}}
  <li>{{firstName}} {{lastName}}</li>
{{/employees}}</ul>
```

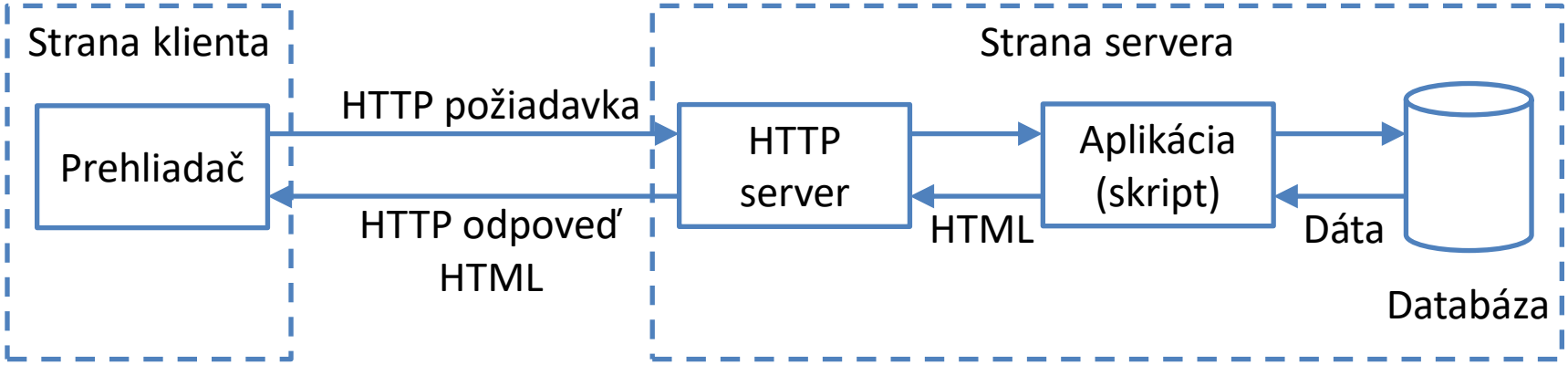
Dynamické generovanie stránok

- Dáta sa dynamicky menia
- Na servery sú uložené najčastejšie v databáze – nie priamo v HTML
- Pre ich zobrazenie je potrebné dynamicky vygenerovať HTML obsah ktorý sa zobrazí v prehliadači

1. Dynamické generovanie stránok na strane servera
2. Dynamické generovanie stránok na strane klienta

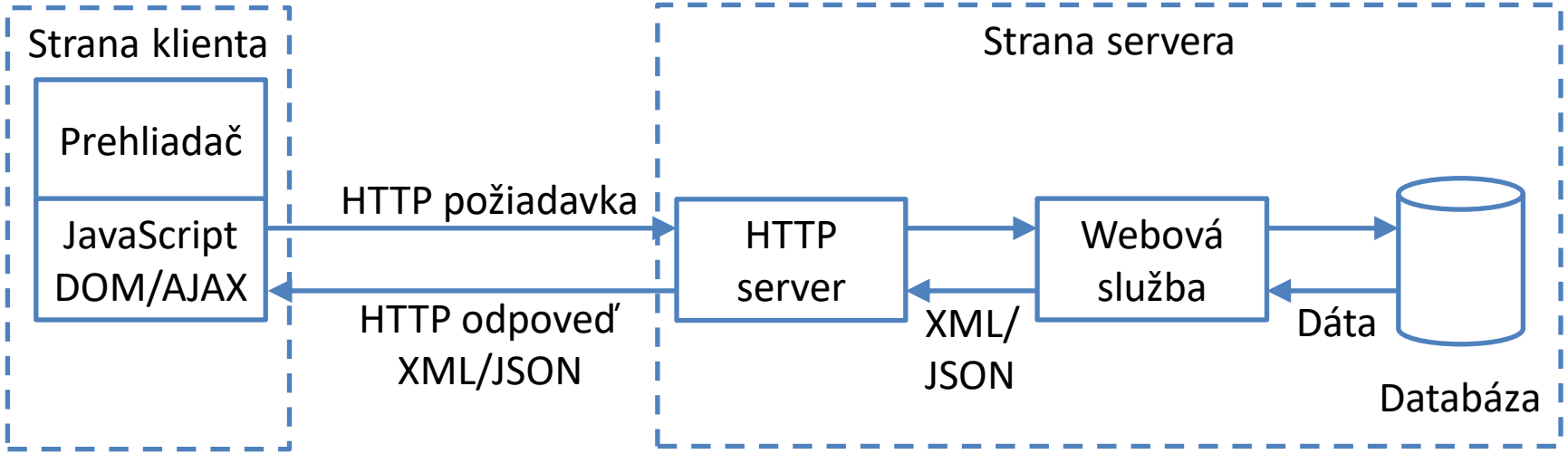
Dynamické generovanie stránok na strane servera

- HTML stránka sa vygeneruje priamo na strane HTTP servera, ktorý ju odošle prehliadaču
- Príklad komunikácie:



Dynamické generovanie stránok na strane klienta

- Na strane servera sú dáta zakódované do formátu pre prenos dát – napr. XML/JSON
- Zakódované údaje sa odošlú na stranu klienta pomocou HTTP
- Prehliadač pomocou JavaScript-u spracuje údaje a dynamicky vygeneruje a zobrazí HTML obsah



AJAX

- Asynchronous JavaScript and XML
- Štandardné rozhranie ktoré umožňuje pomocou JavaScript-u v prehliadači načítať dáta z HTTP servera bez toho aby bolo potrebné znovu načítať celú stránku (tzn. asynchrónne)
  - Pôvodne vzniklo na prenos XML dát ako rozšírenie prehliadača od Microsoftu
  - Nie je však obmedzené na XML formát – ľubovoľný obsah prenášaný pomocou HTTP
  - Dátové formáty: JSON, XML, CSV, ...

JSON

- JavaScript Object Notation
- Textový formát pre zápis štruktúrovaných dát založený na syntaxi JavaScript-u
  - MIME typ: `application/json`, súborová prípona `.json`
  - Nezávislý od programovacieho jazyka
- Umožňuje zápis dátových objektov a ich vlastností:
  - Atomické hodnoty – čísla, reťazce, Boolovské hodnoty, `null`
  - Polia a vnorené objekty

JSON - príklad

```
{
    "firstName": "John",
    "lastName": "Smith",
    "isAlive": true,
    "age": 27,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York"
    },
    "phoneNumbers": ["123 456-789", "234 567-891"],
    "children": [],
    "spouse": null
}
```

Spracovanie JSON v prehliadači

- Objekt `JSON` – štandardné rozhranie JavaScript-u
- Metódy:
  - `JSON.stringify(obj)` – prevedie objekt JavaScript-u na reťazec
  - `JSON.parse(str)` – prevedie reťazec na objekt JavaScript-u
- Príklad načítania dát z reťazca:

```
var str = '{ "firstName": "John", "lastName": "Smith" }';
var obj = JSON.parse(str);
var name = obj.firstName + " " + obj.lastName;
```

Štandardné AJAX rozhranie

- Objekt `XMLHttpRequest` – štandardné AJAX rozhranie JavaScript-u
- Umožňuje prijatie a odosielanie dát
  - HTTP požiadavky GET, POST, PUT, DELETE
- Synchrónne/asynchrónne načítanie dát (tzn. vykonávanie skriptu sa preruší/nepreruší pokiaľ sa dáta nenačítajú po sieti zo servera)
- Základné použitie:
  1. Vytvorenie objektu `XMLHttpRequest`
  2. Zaregistrovanie obslužnej funkcie, ktorá prevezme načítané dáta
  3. Odoslanie HTTP požiadavky

XMLHttpRequest GET príklad (1)

```
function reqListener () {
  // spracovanie odpovede
  console.log(this.responseText);
}

var ajax = new XMLHttpRequest();
ajax.addEventListener("load", reqListener);
ajax.open("GET", "http://www.example.org/data.txt", true);
ajax.send();
```

1. Vytvorenie XMLHttpRequest objektu

XMLHttpRequest GET príklad (2)

```
function reqListener () {
  // spracovanie odpovede
  console.log(this.responseText);
}

var ajax = new XMLHttpRequest();
ajax.addEventListener("load", reqListener);
ajax.open("GET", "http://www.example.org/data.txt", true);
ajax.send();
```

Funkcia ktorá je volaná pri prijatí dát, resp. pri chybe

`this` vo funkcii odkazuje na objekt XMLHttpRequest

2. Zaregistrovanie obslužnej funkcie

XMLHttpRequest GET príklad (3)

```
function reqListener () {
  // spracovanie odpovede
  console.log(this.responseText);
}

var ajax = new XMLHttpRequest();
ajax.addEventListener("load", reqListener);
ajax.open("GET", "http://www.example.org/data.txt", true);
ajax.send();
```

true - asynchrónne volanie, false - synchrónne

3. Vygenerovanie GET požiadavky pre zadanú URL adresu

Vlastnosti objektu AJAX

- Objekt `XMLHttpRequest` umožňuje skontrolovať stav komunikácie so serverom a získať prijaté dáta
- V obslužnej funkcii je prístupný cez `this`
- Stavové a dátové vlastnosti:
  - `status` – stavový kód HTTP, napr.: 200 OK
  - `statusText` – textové stavové hlásenie
  - `responseText` – dáta z odpovede ako reťazec (ak sú dáta zakódované v JSON formáte, je možné použiť `JSON.parse(this.responseText)` pre načítanie dát do objektu)
  - `responseXML` – dáta z odpovede ako DOM objekt (pre XML formát)

XMLHttpRequest POST príklad

```
var data = { "firstName": "John", "lastName": "Smith" };
// ...
function reqListener () {
  if (this.status == 200) {
    alert("Dáta boli úspešne odoslané.");
  } else {
    alert("Došlo k chybe: " + this.statusText);
  }
}
var ajax = new XMLHttpRequest();
ajax.addEventListener("load", reqListener);
ajax.open("POST", "http://www.example.org/data/", true);
ajax.send(JSON.stringify(data));
```

Servisne orientovaná architektúra

- Architektúra pre návrh systémov (aplikácií)
  - Aplikácie sa skladajú z na sebe nezávislých komponentov, ktoré si navzájom poskytujú služby
- Služba
  - Predstavuje jednotku funkcionality: akciu ktorú je potrebné vykonať nad dátami (napr. zápis/čítanie)
  - Používateľ služby nemusí vedieť ako služba vnútorne funguje
- Webová služba
  - Služba poskytovaná pomocou internetových protokolov
  - Použitie služby je ako volanie procedúry na diaľku (tzv. *remote procedure call*) – vstupné parametre, protokol služby, výstupné údaje

Realizácia webových služieb

- SOAP (*Simple Object Access Protocol*)
  - Protokol pre výmenu správ
  - Dáta sú zakódované do XML a prenášane internetovými protokolmi (primárne HTTP)

- RESTful (*Representation State Transfer*)
  - Súbor pravidiel a ohraničení pre implementáciu webových služieb – nie je to len komunikačný protokol

Základné vlastnosti REST

- Vychádzajú z vlastností HTTP ako primárneho komunikačného protokolu
  - Klient-server architektúra
  - Bezstavovosť
  - Dočasne-uložiteľný obsah (*Cacheable*)
  - Viacvrstvový systém
  - Jednotné rozhranie

Jednotné rozhranie REST služieb

- Založené na zdrojoch identifikovaných na webe pomocou URL
  - Zdroje môžu byť navzájom prepojené odkazmi
- Na servery je uložená reprezentácia zdroja jednoznačne definovaná MIME
  - Jeden zdroj môže mať na severy viacero reprezentácií – uložený vo viacerých formátoch
- CRUD (*Create, Read, Update, Delete*)
  - Obmedzený počet operácií, ktoré je možné vykonávať nad zdrojmi a ktoré sú priamo namapované na metódy HTTP
  - Čítanie – GET
  - Vytvorenie – POST
  - Zmena – PUT
  - Zmazanie – DELETE

Príklad REST služby (1)

- Mikroblogovací systém
  - Články, komentáre, tagy
  - Služba dostupná na servery http://wt.kpi.fei.tuke.sk/
- Reprezentácia dát
  - JSON
- Typy zdrojov
  - `article` – článok
  - `comment` – komentár o článku
  - `tag` – kľúčové slová

Príklad REST služby (2)

```
{
  "id": 12345,
  "author": "Peter Bednár",
  "dateCreated": "2018-05-10T08:07:21Z",
  "lastUpdated": "2018-05-10T08:07:21Z",
  "title": "Úvod do jQuery",
  "content": "jQuery je knižnica pre programovanie ...",
  "imageLink": null,
  "tags":["webové technológie", "jquery"]
}
```

Príklad REST služby (3)

- Article
  - GET `/api/article/` – zoznam existujúcich článkov
  - GET `/api/article/{id-článku}` – článok s ID (napr.: `http://wt.kpi.fei.tuke.sk/api/article/12345`)
  - POST `/api/article/` – vytvorenie nového článku
  - PUT `/api/article/{id-článku}` – úprava existujúceho článku
  - DELETE `/api/article/{id-článku}` – mazanie článku

Príklad REST služby (4)

- Comment
  - GET `/api/article/{id-článku}/comment` – zoznam komentárov k článku
  - POST `/api/article/{id-článku}/comment` – vytvorenie nového komentára k článku
  - PUT `/api/article/{id-článku}/comment/{id-komentára}` – úprava existujúceho komentára
  - DELETE `/api/article/{id-článku}/comment/{id-komentára}` – mazanie komentára

Príklad REST služby (5)

- Tag
  - GET `/api/tag` – zoznam kľúčových slov
  - PUT `/api/article/{id-článku}/tag/{kľúčové-slovo}` – pridanie kľúčového slova k článku
  - DELETE `/api/article/{id-článku}/tag/{kľúčové-slovo}` – odstránenie kľúčového slova z článku