



JavaScript a REST

Klientská strana informačního systému

Doc. Ing. Radek Burget, Ph.D.
burgetr@fit.vutbr.cz

Klientská strana IS

- **Tenký klient** – prohlížeč
 - Zobrazuje HTML, odesílá GET a POST požadavky (formuláře)
 - Prezentační logika (chování UI) **na serverové straně**
- **Tlustý klient** – aplikace (webová, mobilní, ...)
 - Volá funkce aplikačního rozhraní (např. REST)
 - Prezentační logika (chování UI) **na klientské straně**

JavaScript

Jazyk a platforma pro tvorbu aplikací

JavaScript

- Původně jazyk pro jednoduché skripty na klientské straně (v prohlížeči Netscape Navigator)
- Postupně vznik rozsáhlejších programů
 - Tlak na rychlost zpracování
- V8 JavaScript engine (Chromium/Chrome), SpiderMonkey (Mozilla)
 - Vykonný JS engine
 - Umožnily tvorbu rozsáhlých aplikací

JavaScript jako platforma

- **Klientský JavaScript**

- Aplikace běží ve webovém prohlížeči
- Prohlížeč poskytuje API
 - DOM, [window](#), atd...

- Samostatný (serverový) JavaScript

- „Osamostatněný“ JavaScript pro tvorbu obecných aplikací
- Stejný jazyk, jiná API, řada dostupných knihoven
- Dnes nejčastěji [node.js](#) a související nástroje

Klientský JavaScript

- Viz [samostatné doplňující slidy](#).
- Prohlížeč není přátelské vývojové prostředí
 - Očekává skripty vložené nebo odkazované z HTML kódu
 - Např. organizace kódu? Knihovny?
- Lokální sestavovací nástroje
 - Opět často `node.js` a doplňující nástroje a knihovny
 - Umožňuje využít existující knihovny a tvořit modulární program
 - Provede *sestavení* do výsledného skriptu (*bundle*), který se vkládá do HTML

Klientské programování

REST klient v JavaScriptu, kdysi též AJAX

Co potřebujeme

- HTML + CSS
 - implementace a design stránky
- DOM
 - změny v dokumentu
- XML, JSON, ...
 - výměna dat
- JavaScript
 - logika UI
- `XMLHttpRequest` nebo `fetch()`
 - JS API v prohlížeči pro odesílání požadavků

Přenášená data

- Strukturovaná data přes HTTP
 - Je třeba *serializace*
 - XML (původně)
 - JSON (odlehčené řešení)
- Části stránky
 - Obvykle přímo útržky HTML

Odesílání dat

- V rámci GET požadavku lze odesílat hodnoty parametrů
 - Standardně klíč-hodnota
`http://www.news.com/articles/?cat=business&id=4827`
 - Někajá konvence zakódování do URL, server musí dekodovat
`http://www.news.com/articles/business/4827`
- V rámci POST, PUT, DELETE lze odeslat libovolná data, je nutno specifikovat MIME typ (hlavička `Content-type` v požadavku)
 - `application/x-www-form-urlencoded` nebo `multipart/form-data` – lze odesílat jen dvojice klíč-hodnota
 - `application/json`, `application/xml`, apod. – přímo serializované strukturované hodnoty

Příjem dat

- Výsledkem požadavku je obvykle odpověď obsahující *serializovaná data*
- Formát serializace je indikován hlavičkou `Content-type` v odpovědi
 - Obvykle opět `application/json` nebo `application/xml`
- Je zvykem, že klient uvede preferovaný formát (i více) pomocí hlavičky `Accept` v požadavku
 - Např. `Accept: application/json`
- Server v rámci svých možností vyhoví nebo vrátí chybu

Stavové kódy

- Stavový kód indikuje výsledek operace
- Provedeno v pořádku
 - `200 OK`
 - `201 Created`
- Chyba
 - `400 Bad request`
 - `401 Unauthorized`
 - `404 Not found`
 - Tělo odpovědi stále může obsahovat data (např. podrobnosti o chybě)

Server

- Implementuje *business logiku* (operace)
 - Každá operace má obvykle svoje URL (případně parametrizovatelné)
- Poskytuje REST API pro jednotlivé operace
- Libovolná serverová platforma
 - Java, .NET, PHP, Python, Node.js, ...
 - Typicky podporováno v různých frameworkcích
- Jednoduchý příklad:
 - [php-rest-db](#) (viz předchozí přednášky)

Klient

- Interakce s uživatelem
 - Zobrazuje data
 - Poskytuje rozhraní pro spouštění business operací a vstup dat
- Volá metody API
 - Vysílá příslušné HTTP požadavky na endpoint URL
- Zobrazuje výsledky volání

Klient v JavaScriptu

- Rozhraní XMLHttpRequest

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && xhr.status == 200) {
        console.log("Response received: " + xhr.responseText);
    }
}
xhr.open("POST", "post-handler.php", true);
xhr.setRequestHeader("Content-type", "application/json");
xhr.send(JSON.stringify(data));
```

Klient pomocí fetch()

```
fetch('https://example.com/profile', {  
  method: 'POST',  
  headers: {  
    'Content-Type': 'application/json',  
  },  
  body: JSON.stringify(data),  
})  
.then(response => response.json())  
.then(data => {  
  console.log('Success:', data);  
})  
.catch((error) => {
```


Klient pomocí fetch() a await

```
async function doPost() {  
  try {  
    var response = await fetch('https://example.com/profile', {  
      method: 'POST',  
      headers: {  
        'Content-Type': 'application/json',  
      },  
      body: JSON.stringify(data),  
    })  
    var data = await response.json();  
    console.log(data);  
  } catch (e) {
```

- viz též asynchronní zpracování, `await` apod.

Jednoduchý klient v jQuery

- `jQuery.ajax(settings)` – asynchronní HTTP požadavek

```
$.ajax({  
  url: "test.html",  
  cache: false  
}).done(function( html ) {  
  $( "#results" ).append( html );  
});
```

- `jQuerygetJSON(url)` – Načte JSON data ze serveru

Klient v jQuery

- jQuery zjednodušuje související úlohy
- Spouštění REST volání (AJAX)
 - Funkce `jQuery.getJSON()`
- Zobrazení dat ve stránce
 - Mnoho možností, např. funkce `append()`
- [Příklad klienta](#)

Same-origin policy

- Bezpečnostní omezení v prohlížečích
- Skript může posílat požadavky jen na zdroje se stejným *origin*
 - Cíl požadavku má stejné **schéma**, **hostname** a **port**, jako má URL skriptu
- Je možno rozvolnit toto omezení na cílovém serveru
 - Server posílá hlavičku `Access-Control-Allow-Origin`, případně další.
 - Např. `Access-Control-Allow-Origin: *`
 - Vhodné pro veřejná API, používat opatrně

REST API s autentizací

- Protokol REST je definován jako *bezstavový*
 - Požadavek musí obsahovat vše, žádné ukládání stavu na serveru
- To teoreticky vylučuje možnost použití sessions pro autentizaci
 - Technicky to ale možné je
 - Sessions s cookies jsou ale použitelné jen v rámci jednoho hostitele
 - Problém např. pro mobilní klienty
- Alternativy pro autentizaci:
 - HTTP Basic autentizace (nutné HTTPS)
 - Použití tokenu validovatelného na serveru – např. JWT
 - Složitější mechanismus, např. OAuth

HTTP Basic

- Standardní mechanismus HTTP, využívá speciální hlavičky
- Požadavek musí obsahovat hlavičku `Authorization`
 - Obsahuje jméno a heslo; nešifrované pouze kódované (base64)
 - **Je nutné použít HTTPS**
 - PHP dekoduje a zpřístupní v `$_SERVER['PHP_AUTH_USER']` a `$_SERVER['PHP_AUTH_PW']`
- Pro nesprávnou nebo chybějící autentizaci server vrací `401 Authorization Required`
 - V hlavičce `WWW-Authenticate` je identifikace oblasti přihlášení
 - Klient tedy zjistí, že je nutná autentizace pro tuto oblast

HTTP Basic v PHP

```
if (!isset($_SERVER['PHP_AUTH_USER'])) {  
    header('WWW-Authenticate: Basic realm="My Realm"');  
    header('HTTP/1.0 401 Unauthorized');  
    echo 'Text to send if user hits Cancel button';  
    exit;  
} else {  
    echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}</p>";  
    echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";  
}
```

JSON Web Token (JWT)

- Řetězec složený ze 3 částí
 1. Header (hlavička) – účel, použité algoritmy (JSON)
 2. Payload (obsah) – JSON data obsahující id uživatele, jeho práva, expiraci apod.
 3. Signature (podpis) – pro ověření, že token nebyl podvržen nebo změněn cestou
- Tyto tři části se kódují (base64) a spojí do jednoho řetězce
 - `xxxxxx.yyyyyy.zzzzzz`

Použití JWT pro autentizaci

1. Klient kontaktuje *autentizační server* a dodá autentizační údaje
 - Stejný server, jaký poskytuje API, nebo i úplně jiný (např. Twitter)
2. Autentizační server vygeneruje podepsaný JWT a vrátí klientovi
3. Klient předá JWT při každém volání API
 - Nejčastěji opět v hlavičce:
`Authorization: Bearer xxxxx.yyyyyy.zzzzzz`
 - API ověří platnost, role uživatele může být přímo v JWT

OAuth

- Požadavek:
 - **Uživatel** má účet na nějaké **službě** (která má API)
 - Chce nějaké **aplikaci** umožnit přístup ke službě přes API
 - Ale nechce své přístupové údaje sdělovat aplikaci
- Protokol OAuth existuje ve dvou verzích:
 - OAuth 1.0 – Složitější, protokolově nezávislý
 - OAuth 2.0 – Zjednodušeno, závisí na HTTPs

OAuth 2 – Postup

- Aplikace musí být předem registrována u Služby
 - Má přidělenou nějakou identifikaci
- Aplikace přesměruje prohlížeč uživatele na autorizační stránku Služby
 - Uživatel provede přihlášení a autorizuje aplikaci
 - Služba přesměruje zpět na aplikaci a předá jí autorizační kód (authorization grant)
- Aplikace pošle POST požadavek obsahující kód na API služby
 - Služba vrátí přístupový token (např. JWT nebo jiný tajný kód)
- Aplikace posílá token s každým požadavkem v hlavičce `Authorization`

OAuth – password grant

- Pokud uživatel může dát přístupové údaje aplikaci
 - Např. aplikace je přímo klient dané služby, mají stejného poskytovatele
 - Aplikace pošle POST požadavek na API obsahující přihlašovací údaje
 - API vrátí přístupový token

Implementace klienta – alternativy

- jQuery je velmi jednoduché, řeší pouze základní úlohy.
- Angular (neplést s Angular.js)
 - Moderní komplexní framework
 - Primárně využívá TypeScript, náročnější nastavení projektu
- React.js
 - Moderní framework, virtuální DOM, vlastní rozšíření JS (JSX)
- Vue.js, ...

Viz např. [SimpleRest@GitHub](#)

Shrnutí

- JavaScript umožňuje tvorbu komplexních klientských aplikací s robustním GUI
- Umožňuje oddělit business logiku (na serveru) od prezentační logiky (na klientovi)
- jQuery usnadňuje nejběžnější úlohy
- Existuje mnoho složitějších frameworků pro rozsáhlé aplikace

A to je vše!

Dotazy?

