HTTP A API

Pokročilý protokol HTTP, úvod do API, REST a dalších příbuzných pojmů včetně praktického využití. Použití videí a zvuku na webu

HTTP

HyperText Transform Protocol je client-server protokol, který při svém vytvoření slouží především k přesunutu hypertextových dokumentů (viz. opakování WBA). Nyní se používá i na jiné věci a nemusí čistě přenášet pouze je.

HTTP dělíme na **požadavky**, které odesílá klient, a **odpovědi**, které zasílá na požadavek server. HTTP je textový protokol a používá konce řádku CRLF (`\r\n`). Níže budeme mluvit o objektech v smyslu požadavku a odpovědi. Objekt je v této části myšleno něco abstraktního, co existuje na serveru a můžeme to upravovat. Budeme taktéž používat výraz **endpoint**, což je spojení metody a URL adresy (resp. její šablony).

Požadavek

Anglicky request, kdy klient se na něco ptá serveru a něco požaduje. Požadavku vždy definujeme:

- **metodu** (method), která určuje, co na daném místě cílu chceme provést, viz. níže
- **cíl** (target), která určuje na jakou URL vlastně cílíme
- **verzi** (version), která určuje verzi HTTP
- **hlavičky** (headers), která určuje různé metadata pro požadavek, např. přihlašovací informace (může být prázdné)
- **tělo** (body), které určuje data, které server zpracuje (může být prázdné)

Metody:

- GET, který slouží pro získání informací o objektu na daném cíli, tělo musí být prázdné
- POST, který slouží k zavolání akce nebo vytvoření objektu na serveru, hodí se na velká data
- PUT, nahraje data na server, pokud však objekt existuje, jen data nahradí
- PATCH, opraví části dat na serveru, pokud objekt existuje
- DELETE, smaže objekt na serveru
- OPTIONS, zjistí, co na daném objektu můžeme provádět za akce
- HEAD, TRACE, CONNECT - různé věci na více, viz. ZDE

Formát HTTP požadavku vypadá následovně:

```
(method) (target) HTTP/(version)\r\n
(header1)\r\n
(header2)\r\n
(...)\r\n
(headern)\r\n
\r\n
(body)\r\n
```

Hlavičky jsou vždy ve formátu `jmeno: hodnota`, kde jméno nesmí obsahovat dvojtečku. Vyplněný požadavek může vypadat:

```
GET /subject/x/selfstudy/prubezne-opakovani HTTP/1.1
User: matej.cajthaml
Host: localhost:8080
Accept: */*
```

Různé hlavičky slouží pro různé věci a tím se budeme zaobírat v dalších stránkách. Hlavičky nejsou omezené.

Odpověď

Anglicky response, kdy server odpovídá na požadavek klienta. Požadavek se zpracuje a vypracuje se odpověď, která se klientovi pošle (např. i odmítnutí). Odpovědi vždy definujeme:

- **statusový kód** (status code), která určuje číslo (100 - 599) toho, co se stalo
- **statusový text** (status text), která určuje pro status code přesnější uvedení
- **verzi** (version), která určuje verzi HTTP
- **hlavičky** (headers), která určuje různé metadata pro odpověď
- **tělo** (body), které určuje data, které zpracuje klient

Statusových kódu je spousta, jejich první číslo však určuje jejich kategorii:

- 1xx — informační (požadavek přijat, pokračuje, ...)
- 2xx — úspěch
- 3xx — přesměrování
- 4xx — chyba požadavku (klienta) nebo nelze zpracovat
- 5xx — chyba serveru (požadavek může být v pořádku)

Je však dobré znát statusové kódy 200, 201, 300, 301, 400, 401, 403, 404 a 500. Existuje spousta stránek pro zobrazování HTTP kódu, pokud máte rádi kočičky, doporučuji tuto stránku ZDE. A pokud ne, tak tuto ZDE.

Formát HTTP odpovědi vypadá následovně:

```
HTTP/(version) (statusCode) (statusText)\r\n
(header1)\r\n
(header2)\r\n
(...)\r\n
(headern)\r\n
\r\n
(body)\r\n
```

A vyplněno vypadá:

```
HTTP/1.1 201 Created
Content-Type: application/text

Vytvoreno! :)
```

Uvnitř body můžeme oběma směry předávat různé data v různých formátech. Důležité je, aby tomu druhá strana rozuměla a mohla data zpracovat. Může tam tedy fungovat i HTML (což je hypertextový dokument) ale i obyčejný text či obrázky.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

VLASTNOSTI POŽADAVKŮ

Pro požadavky by měli platit následující vlastnosti. Jejich realizace však záleží na vytvoření (nastavení) backendu.

- **bezstavové**, jednotlivé požadavky nejsou vázané na sebe a server si nemusí mezi požadavky nic pamatovat
- (některé) jsou **bezpečné**, požadavky pouze čtou data (GET, OPTIONS, ...)
- **uložitelné**, požadavek lze uložit a volat je později či opakovaně
- (některé) jsou **idempotentní**, volání požadavku opakovaně nemění stav serveru (PATCH a PUT)

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

API

Application Programming Interface je komunikační vrstva mezi různými věcmi. Z názvu je nám jasné, že slouží k programovému ovládaní jednotlivých částí. Existují tedy API i na úrovni HW a podobně. V tomto předmětu se budeme zaobírat především **Web API**, které funguje na protokolu HTTP a slouží především jako komunikační vrstva mezi backendem a frontendem.

Web API je velmi rozšířené a používá se skoro úplně všude, a to např. i ve hrách. Často se setkáváme s názvem webové služby, kdy jedno API se chová jako služba, kterou mohou používat tisíce zákazníků a tím si ulehčit práci. Obecně je Web API tedy **balíček URL zdrojů (endpointů) a různých operací**. Často je taktéž nutné se zaregistrovat a získat si přihlašovací kód.

U Web API se setkáme se standardem **REST**, což je architektonický vzor k vytváření moderního API. Když API dodržuje požadavky REST (viz. níže), říkáme, že je **RESTful**. Požadavky jsou následující:

- místo volání různých akcí na serveru, tak klient pracuje s objekty (zdroji) samostatně a je zodpovědný za jejich stav
- každý objekt (zdroj) má svojí vlastní vlastní URL, na které můžeme provádět různé akce, které zjistíme pomocí OPTIONS
- typ práce na objektu určuje jeho metoda
- jeden zdroj může mít více reprezentací (např. umí vracet jak JSON, tak i HTML, viz. pozdější stránky)
- zdroje jsou propojené a uživatelsky čitelné

REST endpoint může tedy vypadat např.:

```
https://api-ssps.cajthaml.eu      // <-- doména
/subject/61253075a238ce375c4ade63 // <-- předmět s id
/grading                          // <-- v předmětu najdeme hodnocení
/user/612539031a6bab58d056b34b    // <-- pro uživatele s id
```

Cread Read Update Delete (**CRUD**) je název, který se používá pro reprezentaci nejčastějších akcí, které na zdrojích používáme.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

FORMÁTY DAT

Při tvorbě API je důležité si uvědomit, v jakém formátu budou data odesílány a přijímány serverem. API se většinou tvoří právě kvůli tomu, aby data mohlo využívat více platforem - a nebyly závislé na jedné věci. V standardu se používá formát XML a JSON, ale zajisté naleznete i jiná API, které používají zcela jiné formáty (CSV, ...). My si tyto formáty vysvětlíme. Velmi často, zejména u velkých API, se stává, že používají více formátů a stačí si o ně požádat pomocí hlaviček.

Jako základ si musíme vysvětlit ještě formát, který vlastně není formát - a to obyčejný text. Tyto data jsou špatně čitelná, protože si každý může určit jakýkoliv formát a ten může být popsán v nějaké dokumentaci. Obecně je nutné si napsat vlastní parser a složitě nad vlastním formátem přemýšlet a proto se prakticky nepoužívá. Jako textový, obyčejný, formát považujeme CSV.

XML

Extensible Markup Language je formát, který byl v začátcích určen pro uchovávání a přenášení dat. Jedná se o značkovací a velmi upovídaný formát. Všechny data musí být obalená v značkách se jménem. XML je velmi podobné HTML, a není to náhoda, protože XML je rodič HTML. HTML však určuje jednotlivé tagy a atributy, včetně jejich vykreslování a chování. Taktéž dovoluje nepárové tagy. XML je nepodporuje a nemá definovaný ani jeden tag a atribut - můžeme je používat jak chceme a to často je problém.

Formáty se velmi liší a spousta lidí má pouze jeden tag, a obsah mají v daném tagu uvnitř jako dítě, či jenom atributy. Někdo má pro každý typ dat vlastní tag. Nevýhody XML již byly zmíněné - je velmi výřečný a použití je velmi rozmanité, což může způsobovat problémy. Výhoda je např. to, že je velmi dobře čitelný strojem i uživatelem, a že lze používat na spoustu věcí. Ukázka:

```
<chapter name="Transportation">
   <dataList name="Air" rank="45" column="2">
       <data name="Registered air carriers" value="4"/>
       <data name="Aircraft registered" value="48"/>
       <data name="Annual passenger traffic" value="5727200"/>
       <data name="Freight traffic" value="25230000"/>
       <data name="Airports" rank="45" value="128"/>
       <data name="Paved runways" value="41"/>
       <data name="Unpaved runways" value="87"/>
       <data name="Heliports" value="1"/>
   </dataList>

   <dataList name="Pipelines" column="2">
       <data name="Gas" unit="km" value="7160"/>
       <data name="Oil" unit="km" value="675"/>
       <data name="Refined products" unit="km" value="94"/>
   </dataList>
   <dataList name="Railways" rank="24" column="2">
       <data name="Total" unit="km" value="9408"/>
       <data name="Standard gauge" unit="km" value="9385"/>
       <data name="Narrow gauge" unit="km" value="23"/>
   </dataList>
   <dataList name="Roadways" rank="82" column="2">
       <data name="Total" unit="km" value="55744"/>
       <data name="Paved" unit="km" value="55744"/>
       <data name="Expressways" unit="km" value="1252"/>
   </dataList>
```

```
    <dataList name="Waterways" rank="76" column="2">
        <data name="Total" unit="km" value="664"/>
    </dataList>

    <dataList name="Parties">
        <data name="Christian Democratic Union-Czechoslovak People's Party">
            <additional name="Short">KDU-CSL</additional>
            <additional name="Leader">Pavel BELOBRADEK</additional>
        </data>
        <data name="Civic Democratic Party">
            <additional name="Short">ODS</additional>
            <additional name="Leader">Petr FIALA</additional>
        </data>
        <data name="Communist Party of Bohemia and Moravia">
            <additional name="Short">KSCM</additional>
            <additional name="Leader">Vojtech FILIP</additional>
        </data>
        <data name="Czech Social Democratic Party">
            <additional name="Short">CSSD</additional>
            <additional name="Leader">Jan HAMACEK</additional>
        </data>
        <data name="Freedom and Direct Democracy">
            <additional name="Short">SPD</additional>
            <additional name="Leader">Tomio OKAMURA</additional>
        </data>
        <data name="Green Party">
            <additional name="Short">SZ</additional>
            <additional name="Leader">Petr STEPANEK</additional>
        </data>
        <data name="Mayors and Independents">
            <additional name="Short">STAN</additional>
            <additional name="Leader">Petr GAZDIK</additional>
        </data>
        <data name="Movement of Dissatisfied Citizens">
            <additional name="Short">ANO</additional>
            <additional name="Leader">Andrej BABIS</additional>
        </data>
        <data name="Party of Civic Rights">
            <additional name="Short">SPO</additional>
            <additional name="Leader">Lubomir NECAS</additional>
        </data>
        <data name="Pirate Party">
            <additional name="Short">Pirates</additional>
            <additional name="Leader">Ivan BARTOS</additional>
        </data>
        <data name="Tradition Responsibility Prosperity 09">
            <additional name="Short">TOP 09</additional>
            <additional name="Leader">Jiri POSPISIL</additional>
        </data>
    </dataList>

</chapter>
```

JSON

JavaScript Object Notation je velmi oblíbený způsob přenášení dat, i když lze použít jako uchovávání dat. Jeho hlavní důvod použití je to, že je velmi přehledný, krátký a není zbytečně výřečný. Je velmi podobný tomu, jak se zapisují objekty v JavaScriptu. Podporuje následující typy:

- číslo
- textový řetězec

- objekt
- pole
- null

Oproti JS objektům se v JSON klíče zapisují vždy v dvojitých uvozovkách (jednoduché se vůbec nepoužívají). Mezi nevýhody JSONu patří to, že je velmi syntakticky striktní (více, než samotné JS) a nepodporuje komentáře. Ukázka:

```
{
    "meta": {
        "status": "success",
        "requested": "2022-02-20T15:22:57.887Z"
    },
    "data": {
        "categories": [
            {
                "_id": "61253e35fcddea8da5d4350f",
                "name": "Zkoušení",
                "description": "Písemné a ústní zkoušení",
                "position": 4,
                "required": 100
            },
            {
                "_id": "61253e45fcddea8da5d43517",
                "name": "Aktivita",
                "description": "Aktivita v hodině a mimo ní",
                "position": 3,
                "required": 0
            },
            {
                "_id": "61253e9ffcddea8da5d43529",
                "name": "Práce",
                "description": "Práce v hodině a doma",
                "position": 2,
                "required": 50
            },
            {
                "_id": "61e86e481b41cf472985203c",
                "name": "Závěrečná práce",
                "description": "\"Náhodné\" povídání o závěrečné práci?",
                "position": 1,
                "required": 0
            }
        ]
    }
}
```

Pro tyto výše uvedené formáty existují knihovny a nebo si musíme pro jejich práci napsat vlastní. Spousta jazyků má již automaticky připravené nástroje pro používání těchto dvou formátů.

Když umí API přenášet XML pomocí HTTP, tak automaticky umí přenášet i HTML. Jak jsme si řekli, HTML jen modifikuje a definuje určité tagy a atributy.

Všechny formáty se používají na obousměrnou komunikaci mezi SERVEREM ↔ CLIENTEM, a nebo může být např. jedna cesta definována jinak. Vždy je důležité, aby API uvádělo, čemu rozumí, a formát dat se naznačoval v hlavičce (nejčastěji se používá Content-Type např. s hodnotou text/plain, `application/json` či `text/html`)

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

VERZOVÁNÍ API

Důležitou součásti API jsou jeho verze. Verze se používají tehdy, když se API za svého života mění a to tak, že změny mohou způsobit katastrofální selhání. Na API jsou závislé různé služby a při změně by mohlo dojít k rozbití dané služby - a to samozřejmě nechceme. Proto jakékoliv změny (oprava chyb se většinou dělá rovnou!) ukládáme do verzí, které číslujeme vzestupně. Při změně verze vždy oznámíme změnu (např. v dokumentaci, e-mailem, ...) a určíme datum, kdy daná verze API přestane být podporovaná a bude ukončena.

Co ale dělat s daty, které již nechceme publikovat? Pokud je chceme smazat okamžitě, jednoduše z daného endpointu budeme vracet, že dané zdroje neexistují a při např. jeho seznamu, vrátíme prázdné pole. Nikdy není dobrý nápad smazat endpoint bez notifikace a zvětšení verze.

Jednotlivé verze většinou určujeme na začátku URL adresy, např. tedy: `/api/v3/user/matej.cajthaml`, kde v3 značí třetí verzi. Za běhu API se může stát, že těchto verzí API poběží více, při velkých API se často stává, že mají po vydání nové verze např. rok svojí životnost, aby se rozbilo co nejméně služeb závislých.

Pracujeme-li však na API, které používá například jenom jedna služba, a nebo je pouze interní a týmy pracují rychle, tyto verze vynecháváme. To samé děláme např. při začátcích developmentu, až bude vše hotové, zabalíme ji do jedné (první) verze.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

AJAX

Asynchronous JavaScript And XML je celkem starý pojem, který referuje na to, že místo toho, aby všechen stav stránky byl určen při jednom GET požadavku na server, tak daná stránka si za svého běhu pomocí JS získává data pro zobrazení voláním na server (resp. API).

Stránka se tedy minimálně refreshuje a první požadavek je většinou bez dat, které se doplní až pomocí JS. AJAX, resp. jeho odnože, se používají na moderních stránkách vždy. Mezi hlavní výhody patří právě to, že nemusíme refreshovat stránku, můžeme stahovat data bez nutnosti uživatele něco dělat (např. chatové upozornění, ...). Mezi nevýhody patří např. to, že často musíme právě stránky vyplňovat daty stejně na serveru díky SEO, protože bez toho by vyhledávače nemohli vidět data a ty si dát do indexu (velmi zjednodušeně).

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

NÁSTROJE

Na HTTP je dobré mít připravený nějaký nástroj:

- Postman
- Insomnia
- https://reqbin.com
- a další

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

EXPRESS

Express je nejpopulárnější framework pro Node.js. Je jednoduchý na použití, ale poskytuje řešení pro různé problémy, jako je routování, middleware, a práce se šablonami. Express má také mnoho pluginů, které rozšiřují jeho funkčnost.

Instalace a Použití

Pro použití Express stačí nainstalovat balíček `express` do projektu, vytvořit instanci aplikace, a nastavit aplikaci, aby naslouchala na určitém portu. Tento port se pak používá v aplikacích pro komunikaci s backendem.

```
import express from 'express';

const app = express();

app.listen(3000, () => {
    console.log('Server is running on http://localhost:3000');
});
```

Middleware

Middleware v Expressu jsou funkce, které mají přístup k požadavku (req), odpovědi (res), a funkci next. Middleware může provést nějakou akci a předat požadavek dalšímu middleware v pořadí. Middleware lze použít globálně pro celou aplikaci, nebo jen pro konkrétní endpointy.

```
app.use((req, res, next) => {
  console.log('Request received');
  next();
});
```

Middleware může být také použito pro omezení přístupu, práci s CORS, a zpracování vstupních dat.

Vstup

Vstupní data mohou být čtena z těla požadavku, URL (query parametry a cesta), nebo hlaviček. Pro zpracování těchto dat se často používá middleware jako `body-parser`, který umožňuje jednoduše pracovat s různými formáty dat.

Validace

Pro validaci vstupních dat se používají knihovny jako `express-validator` nebo `joi`. Validace je důležitá, protože vstupní data mohou být nedůvěryhodná a mohou obsahovat chyby nebo škodlivý kód.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

DATABÁZE

Existují různé typy databází, jako jsou relační databáze (SQL) a dokumentové databáze (NoSQL). Relační databáze ukládají data v tabulkách, které mají řádky a sloupce. Každá tabulka má primární klíč, který jednoznačně identifikuje každý řádek, a cizí klíče, které odkazují na jiné tabulky.

SQL

SQL (Structured Query Language) je jazyk pro práci s relačními databázemi. Pomocí SQL můžeme vytvářet tabulky, vkládat data, vybírat data, aktualizovat data, a mazat data.

Použití DB na Backend

V Node.js je mnoho knihoven pro práci s databázemi, jako je `pg` pro PostgreSQL nebo ORM (Object-Relational Mapping) nástroje jako `sequelize` a `prisma`. Tyto nástroje usnadňují práci s databázemi a poskytují bezpečné způsoby pro práci s daty.

Připojení k DB

```
const { Client } = require('pg');
const client = new Client({
  connectionString: 'postgresql://user:password@localhost:5432/mydb'
});
client.connect();
```

Výběr Dat

```
client.query('SELECT * FROM users', (err, res) => {
  console.log(res.rows);
  client.end();
});
```

SQL Injection

SQL Injection je bezpečnostní chyba, kdy útočník může vložit vlastní SQL kód do parametru. Tomu se lze vyhnout použitím správných metod pro práci s databází, jako jsou prepared statements.

```
const text = 'SELECT * FROM users WHERE id = $1';
const values = [1];
client.query(text, values, (err, res) => {
  console.log(res.rows);
  client.end();
});
```

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.

REVERZÍ PROXY

HTTP používá port 80 a HTTPS port 443. Na jednom PC může běžet více serverů, pokud používají různé porty. Reverzní proxy je server, který přesměrovává požadavky na jiný server nebo port, což se často používá pro zajištění bezpečnosti, load balancingu a škálování.

Poznámky:

Žádné poznámky k této lekci jste ještě nezapsali.