



Webová aplikační rozhraní

Volání aplikační logiky pomocí HTTP

doc. Ing. Radek Burget, Ph.D.

burgetr@fit.vutbr.cz

Webová API

- Na HTTP požadavek můžeme chápat jako na **volání vzdálené funkce na serveru**
 - Předáme jméno a parametry
 - Server vykoná nějaké operace
 - Vráťí výsledek (tradičně v HTML)
- Serializace umožňuje předávat strukturovaná data
 - V HTTP požadavku – např. POST
 - V HTTP odpovědi – v těle odpovědi (dokument)
- Nabízí se využití zejména XML a JSON

Standardy pro webová API

- **XML-RPC** (1998)
 - Využívá pouze HTTP POST požadavky, přenáší se XML (**Content-Type: text/xml**)
 - Definuje jednoduchý formát dokumentů pro popis volání a odpovědi
 - Jméno volané funkce, jména parametrů
 - Výsledek volání (návratové hodnoty)
- **SOAP** (Simple Object Access Protocol, 2000) – **Web Services**
 - Rozšíření formátu zpráv
 - WSDL (*Web Service Description Language*)
 - Umožňuje popsat a sdílet rozhraní služby v XML
 - Možnost automatického generování klienta na implementační platformě

Příklad SOAP

```
<env:Envelope xmlns="...">
  <env:Header/>
  <env:Body>
    <m:jePrvocislo xmlns:m="urn:mojeURI">
      <cislo xsi:type="xs:long">1987</cislo>
    </m:jePrvocislo>
  </env:Body>
</env:Envelope>
```

```
<env:Envelope xmlns="...">
  <env:Body>
    <m:jePrvocisloResponse xmlns:m="urn:mojeURI">
      <return xsi:type="xsd:boolean">true</return>
    </m:jePrvocisloResponse>
  </env:Body>
</env:Envelope>
```

REST

- XML-RPC a SOAP jsou (teoreticky) nezávislé na HTTP
- Vše se popisuje v XML dokumentech
 - Rozsáhlý standard, značná složitost dokumentů
 - Obtížná implementace
- REST – Representational state transfer
 - Využijeme co nejvíce vlastnosti HTTP
 - Zjednodušíme přenášená data

GraphQL

- <https://graphql.org/>
- Motivace: klient (klienti) potřebují v různých situacích různá data
 - Např. stránka „seznam osob“ vs. „detail osoby“
- REST endpoint vrací vždy stejnou strukturu
 - Redundance dat (nevyužijeme všechna data)
 - Více dotazů ((ne)efektivita, složitější logika klienta)
- Řešení GraphQL
 - Popis datového modelu API (na serveru, speciální jazyk)
 - Dotaz na API specifikuje požadovaný tvar odpovědi

Dotazování GraphQL

- Jediné endpoint URL
- Odeslání přes GET
 - `http://myapi/graphql?query={me{name}}`
- Odeslání přes POST
 - Data `application/json`
`{"query": "{me{name}}"}"`
 - Data `application/graphql`
`{me{name}}`

REST

Nejjednodušší cesta k webovému API

REST

- Předpokládá CRUD (Create-Retrieve-Update-Delete) operace s *entitami*
 - Ale ve skutečnosti přistupujeme k business vrstvě, ne přímo k datům!
 - Tzn. voláme aplikační logiku
- Úzká vazba na HTTP
 - Využití HTTP metod a jejich významu
 - Využití stavových kódů v HTTP
- Nedefinuje formát přenosu dat, obvykle JSON, méně XML (často obojí)

Endpoints

- Endpoint = URL, na které lze zaslat požadavek
- Reprezentuje **zdroj** (*resource*), který má nějaký **stav** (*state*)
- Endpointy pro operace se zdroji
 - Kolekce entit, např. <http://obchod.cz/api/objednavky>
 - Jedna entita, např. <http://obchod.cz/api/objednavky/8235>
- Endpointy pro volání funkcí
 - Např. <http://obchod.cz/api/odesli-objednavku>

Metody HTTP – Operace se zdroji

- **GET**
 - Čtení stavu zdroje (read)
- **POST**
 - Přidání podřízeného zdroje (přidání do kolekce, create)
- **PUT**
 - Nahrazení zdroje novým stavem (update)
- **PATCH**
 - Nahrazení části zdroje (update)
- **DELETE**
 - Smazání zdroje (delete)

Metody HTTP – Volání funkcí

- **GET i POST**

- Vykoná operaci vrátí výsledek (serializovaná data)
- Pokud je výsledkem operace nový zdroj, jeho URL se vrátí v hlavičce **Location**.

Stavový kód

- Stavový kód odpovědi HTTP může odpovídat výsledku operace
- Typicky například:
 - 200 Ok
 - 201 Created
 - 400 Bad request
 - 403 Forbidden
 - 404 Not found
 - 500 Internal server error

REST v PHP

Co je třeba k implementaci REST v PHP?

Co musíme řešit?

1. Analýzu HTTP požadavku

- Metoda a cesta → **routing**

2. Dekódování vstupních dat

- Parsování json, dekodování GET parametrů → **deserializace**

3. Vykonání příslušné operace

- Aplikační logika

4. Zakódování a odeslání výsledku

- Vytvoření JSON, a odeslání → **serializace**

Vrstvy aplikace - serverové GUI

Vrstvy aplikace - využití API

REST v čistém PHP

- Vyhodnocení metody a cesty požadavku
 - `$_SERVER['REQUEST_METHOD']` a `$_SERVER['PATH_INFO']`
- Přečtení těla požadavku (pokd je)
 - `file_get_contents('php://input')`
- Nastavení stavového kódu a hlaviček odpovědi
 - `http_response_code(), header()`
- Serializace a deserializace
 - `json_encode(), json_decode()`

Demo: <https://github.com/DIFS-Teaching/basic-demos/tree/master/php-rest-db>

Využití PHP frameworku

- Webový framework zajistí routing a zavolání *controlleru*
- Controller volá aplikační logiku a definuje odpověď
- Dekódování JSON:
 - Podpora deserializace. Např. `$request->json()->all()` v Laravel.
- Odeslání JSON:
 - [Symfony](#)
 - [Laravel](#)
 - [Nette](#)

Pokročilejší řešení

- Framework může automatizovat vše kromě aplikační logiky
- Ta může mít podobu funkce se standardními parametry a návratovou hodnotou
- Příklady:
 - Java (JAX-RS)
<https://github.com/DIFS-Teaching/jsf-basic/blob/master/src/main/java/org/fit/pis/api/People.java>
 - .NET core
<https://dotnet.microsoft.com/apps/aspnet/apis>

A to je vše!

Dotazy?

