



REST

Architektúra pre lepšie webové aplikácie a webové služby

Peter Rybár



Obsah

- REST- Architektúra pre lepšie webové aplikácie a služby
- Vyberme si REST web framework
- Webová aplikácia ako klient ku RESTovským webovým službám, AJAX
- Zhrnutie



REST

Architektúra pre lepšie
webové aplikácie a služby

Peter Rybár

WEB

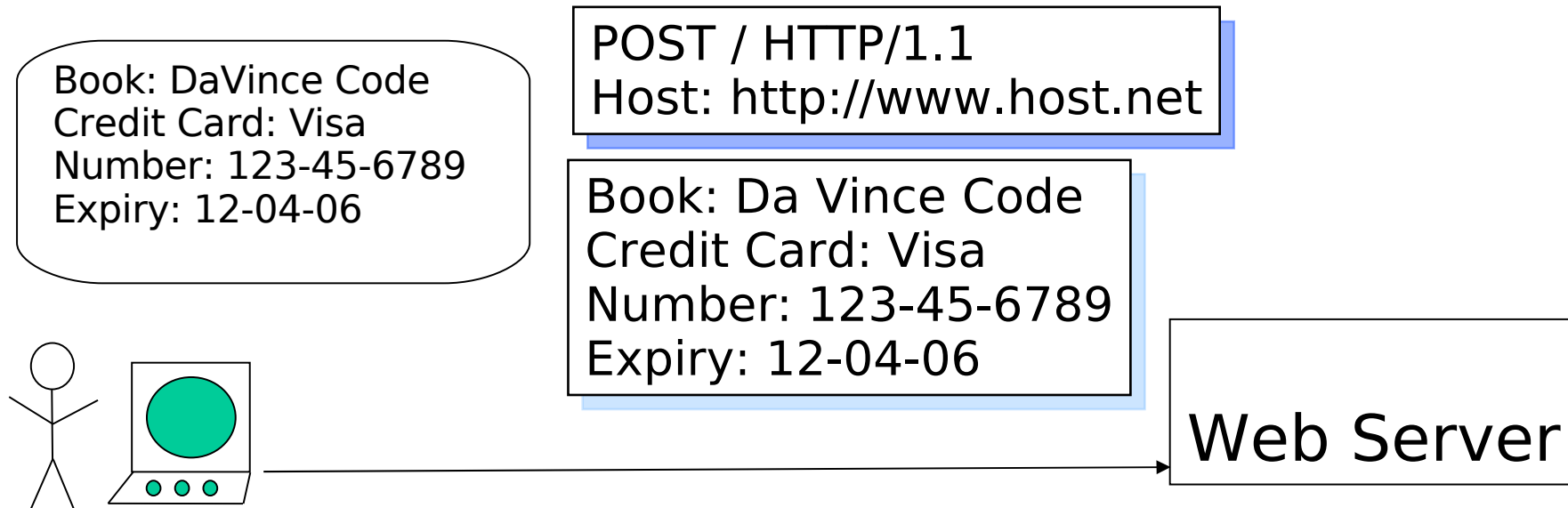
- **World Wide Web** (skrátene **Web**) je systém vzájomne nalinkovaných hypertextových dokumentov prístupných prostredníctvom **HTTP** v sieti Internet
- **World Wide Web** vytvorili Sir **Tim Berners-Lee** and **Robert Cailliau** v roku **1989** v **CERNe** v Ženeve, Švajčiarsko
- Používateľ Webu (HTTP klient) prezerá web stránky obsahujúce **text, obrázky, videá**, a iné **multimédiá** a naviguje sa medzi nimi pomocou **hyperliniek** v **HTML**

Web: Použitie HTTP GET



- Klient zadá do svojho prehliadača: **http://www.host.net**
- Prehliadač vytvorí **HTTP hlavičku**
 - HTTP hlavička identifikuje:
 - Požadovanú **akciu**: GET ("get me resource")
 - Cieľový **stroj** (www.host.net)

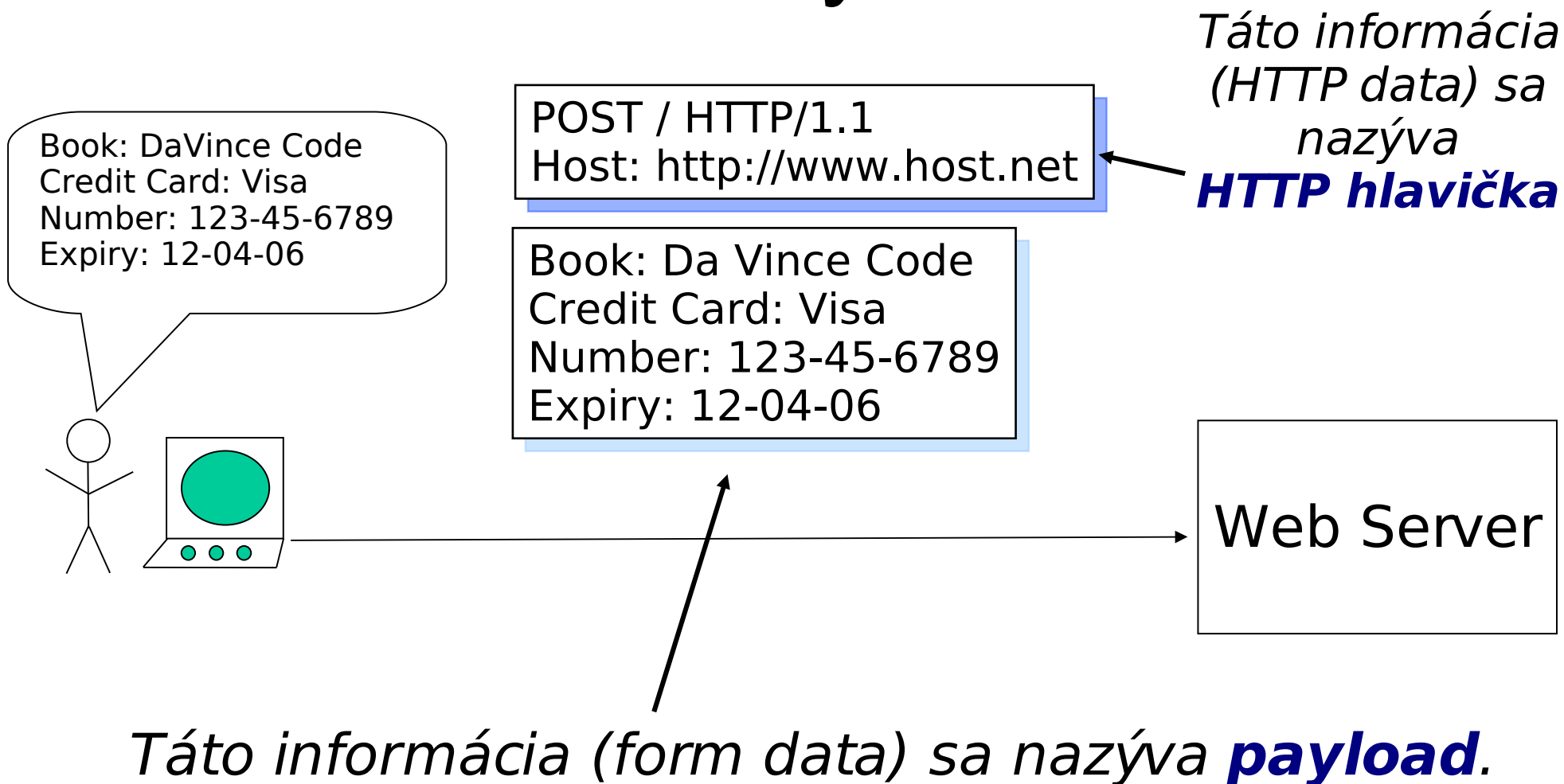
Web: Použitie HTTP POST



- Klient vyplní **formulár** na webovskej stránke
- Prehliadač vytvorí **HTTP hlavičku** a dáta z formulára umiestni do **HTTP body**
- **HTTP hlavička** identifikuje:
 - Požadovanú **akciu**: POST
 - Cieľový **server** (host.net)
- **HTTP body** obsahuje:
 - **Data** POSTnute (encoded form data)

Web: Terminológia

Hlavička and Payload



Web: Operácie HTTP API

- HTTP poskytuje jednoduchú množinu operácií.
- Všetok Web prenos je realizovaný prostredníctvom jednoduchého **HTTP API**:
 - **GET** = "daj mi nejakú informáciu" (Retrieve)
 - **POST** = "tu máš nejakú novú informáciu" (Create)
 - **PUT** = "tu je nejaká update infomácia" (Update)
 - **DELETE** = "vymaž nejakú informáciu" (Delete)
- **HTTP API** je **CRUD** (Create, Retrieve, Update, a Delete)

Webové komponenty

■ Firewally:

- Rozhodujú ktoré HTTP správy (messages) môžu von a ktoré môžu dnu
- Uplatňujú **Webovú bezpečnosť**

■ Route:

- Rozhodujú kde poslať HTTP správy (messages)
- Riadia **Webovú škálovateľnosť**

■ Caches:

- Rozhodujú či uložená kópia HTTP správy (message) môže byť použitá
- Zvyšujú **Webovú rýchlosť**

Webové komponenty: Cache



- Cache sa rozhoduje, či by mala použiť ešte platnú kópiu HTTP správy (dokumentu), alebo si vyžiadať aktuálnejšiu správu od servera na základe HTTP hlavičky
- Všetky rozhodnutia sú na základe **HTTP hlavičky**
- **Cache nikdy nepozerá do prenášaných dát HTTP správ** (payload)

Webové komponenty: Analógia s poštou

- Prečo sa Webové komponenty rozhodujú výhradne na základe informácií z hlavičky?
- Pošta triedi listové zásielky a balíky na základe informácií uvedených v hlavičke obálky (balíka)
- **Nikto** na pošte **nie je oprávnený pozrieť sa dovnútra listu alebo balíka**
- Každý pokus prezerať obsah zásielky sa považuje za porušenie listového tajomstva, čo je **porušenie zákona**

Webové služby

- Softvérový systém navrhnutý pre komunikáciu Stroj-Stroj prostredníctvom siete internet.
- Sada nástrojov, ktoré môžu byť použité rôznym spôsobom na rôzne účely.
- Tri najčastejšie spôsoby použitia sú:
 - **RPC**
 - **SOA**
 - **REST**

Webové služby: RPC

- Prvé nástroje pre Webové služby boli zamerané na RPC
- XML-RPC vytvoril Dave Winer v roku 1998 (Microsoft)
- RPC Webové služby reprezentujú **interface pre volanie vzdialenej funkcie** (metódy)
- Veľmi jednoduchý protokol, definujúci **iba základné dátové typy a príkazy**

Webové služby: RPC

Príklad XML-RPC

POST /RPC HTTP/1.0
Host: example.org
Content-Type: text/xml
Content-length: nnn

```
<?xml version="1.0"?>  
<methodCall>  
  <methodName>examples.getStateName</methodName>  
  <params>  
    <param>  
      <value><i4>41</i4></value>  
    </param>  
  </params>  
</methodCall>
```

Webové služby: RPC

- Niektorí vývojári stále preferujú XML-RPC pred SOAP pre jeho **minimalizmus a jednoduchosť použitia**
- Sú **kritizované pre silnú väzbu**, pretože boli implementované *mapovaním služieb priamo do špecifických funkcií a metód daného jazyka*
- Keď bola do XML-RPC zavedená nová funkcionálna, bol uvedený nový štandard známy dnes ako SOAP

Webové služby: SOA

- Základnou jednotkou komunikácie je **správa (message)**, skôr než procedúra
- Tejto architektúre zvykne byť označovaná ako **“message oriented”**
- Webové služby môžu byť použité na implementáciu konceptu **servisne orientovanej architektúry (SOA)**

Webové služby: SOA

- Najčastejšie používaný protokol pre SOA je SOAP Document Literal
- SOAP pôvodne ako **Simple Object Access Protocol**
- Neskôr ako **Service Oriented Architecture Protocol**
- SOAP pôvodne navrhli Dave Winer, Don Box, Bob Atkinson, a Mohsen Al-Ghosein v roku 1998, v pozadí s firmou Microsoft

Webové služby: SOA

Príklad SOAP Document Literal

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
```

```
<soap:Envelope
```

```
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
  <soap:Body xmlns:m="http://www.example.org/stock">
```

```
    <m:GetStockPrice>
```

```
      <m:StockName>IBM</m:StockName>
```

```
    </m:GetStockPrice>
```

```
  </soap:Body>
```

```
</soap:Envelope>
```

Webové služby: REST

- **REST** - **Representational state transfer**
- RESTovské Webové služby sa pokúšajú emulovať HTTP a podobné protokoly obmedzením rozhrania na sadu štandardných operácií (ako, GET, PUT, POST, DELETE).
- **Dôraz** sa kladie **na interakciu so stavovými zdrojmi** (resources), než na správy (messages) alebo operácie (procedúry)

Webové služby: REST

Príklad REST

POST /parts/12345 HTTP/1.0
Host: example.org
Content-Type: text/xml
Content-length: nnn

```
<?xml version="1.0"?>  
<p:Part xmlns:p="http://www.parts-depot.com">  
  <ID>00345</ID>  
  <UnitCost currency="USD">0.10</UnitCost>  
  <Quantity>10</Quantity>  
</p:Part>
```

Webové služby: SOAP vs REST

SOAP

The following is a sample SOAP request and response. The [placeholders](#) shown need to be replaced with actual values.

```
POST /DemoWebServices2.8/service.asmx HTTP/1.1
Host: api.efxnow.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "https://api.efxnow.com/webservices2.3/GetTime"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetTime xmlns="https://api.efxnow.com/webservices2.3" />
  </soap:Body>
</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetTimeResponse xmlns="https://api.efxnow.com/webservices2.3">
      <GetTimeResult>string</GetTimeResult>
    </GetTimeResponse>
  </soap:Body>
</soap:Envelope>
```

Webové služby: SOAP vs REST

HTTP GET

The following is a sample HTTP GET request and response. The [placeholders](#) shown need to be replaced with actual values.

```
GET /DemoWebServices2.8/service.asmx/GetTime? HTTP/1.1
Host: api.efxnow.com
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="https://api.efxnow.com/webservices2.3">string</string>
```

HTTP POST

The following is a sample HTTP POST request and response. The [placeholders](#) shown need to be replaced with actual values.

```
POST /DemoWebServices2.8/service.asmx/GetTime HTTP/1.1
Host: api.efxnow.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="https://api.efxnow.com/webservices2.3">string</string>
```

REST

- Representational State Transfer je **štýl softvérovej architektúry** určený pre **distribuované hypermediálne systémy** ako je **World Wide Web**

REST: Pojem REST

- REST – pojem zaviedol **Roy Fielding** v jeho **Ph.D. dizertačnej práci**^[1] pre **popis architektonického štýlu sieťou prepojených systémov**
- REST je akronym ktorého význam je **Representational State Transfer**
- REST striktne definuje kolekciu princípov sieťovej architektúry, ktorá **vysvetľuje ako sú zdroje (resources) definované a adresované**

[1] <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

REST: Poslanie

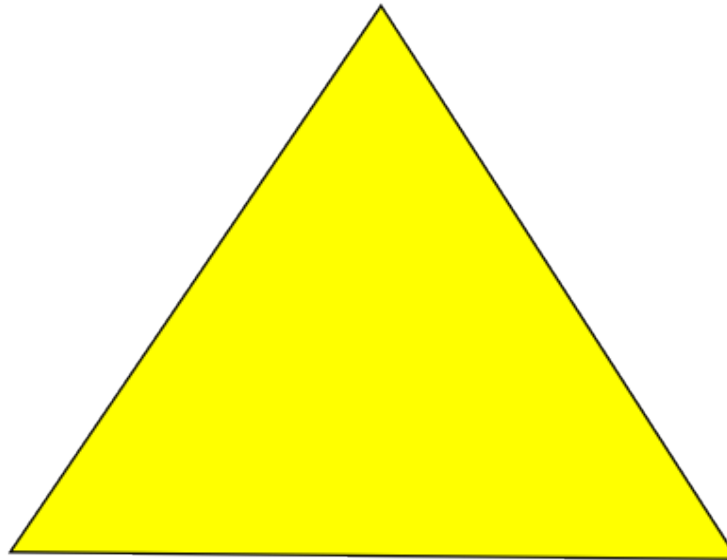
- REST je učený aby uviedol spôsob, **ako sa má správať dobre navrhnutá webová aplikácia**:
 - sieť web **stránok, virtuálnych stavových strojov**
 - kde užívateľ prechádza cez aplikáciu prostredníctvom **liniek, prenosov stavu**
 - výsledkom čoho je nasledujúca stránka, **reprezentujúca** nasledujúci **stav aplikácie**, prenesená k užívateľovi

REST: restovský trojuholník

Nouns

(Unconstrained)

eg <http://wikipedia.org/>



Verbs

(Constrained)

eg GET

Content Types

(Constrained)

eg HTML

REST: restovský trojuholník

- Klient referencuje **zdroj** pomocou URL
- Klientovi je vrátená **reprezentácia** zdroja
- Reprezentácia uvedie klienta do nového **stavu**
- Klient zvolí hyperlinku ďalšieho **zdroja**
- Klientová aplikácia teda robí transfér stavu s každou reprezentáciou zdroja
- **Nová reprezentácia – nový stav!**



REST: Motivácia

- Zozbierať charakteristiky webu, ktoré ho spravili úspešným
- Následne použiť tieto charakteristiky ako návod pre evolúciu webu

REST: Arch. štýl, nie štandard

- **REST nie je štandard!**
- Neuvidíte W3C špecifikáciu RESTu
- Nemali by ste vidieť predávať developerské toolkity od IBM, Microsoft alebo Sunu
- **Prečo?**
- Lebo **REST je iba architektonický štýl** (analógia s klient-server štýlom)
- Môžete RESTu iba rozumieť a dizajnovat' aplikácie podľa neho

REST: Arch. štýl, nie štandard

- Hoci **REST** nie je štandard, používa štandardy:
 - **HTTP**
 - **URL**
 - **XML/HTML/GIF/JPEG/...** (Resource Representations)
 - **text/xml, text/html, image/gif, image/jpeg, ...** (MIME Types)

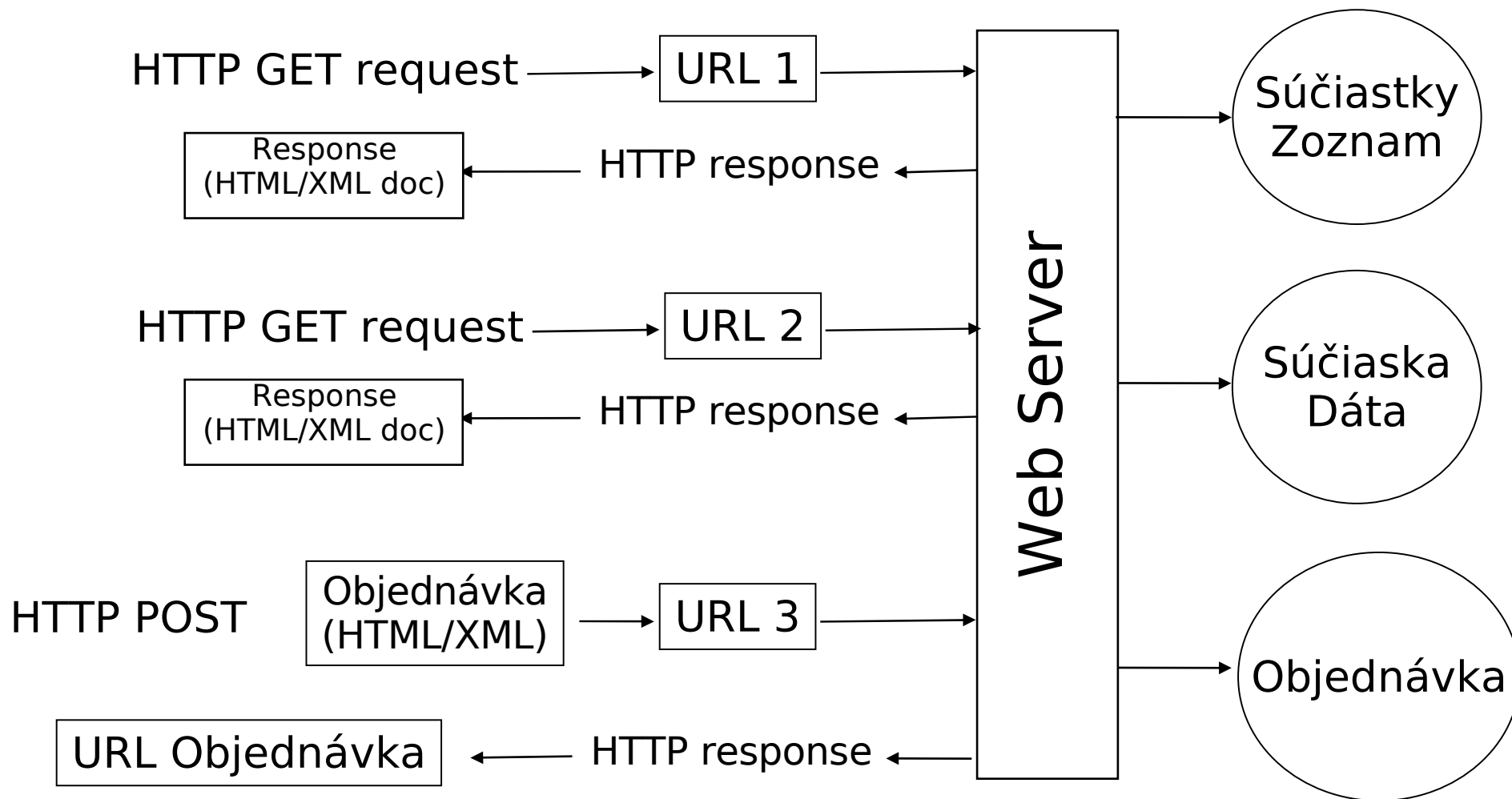
REST: Web je REST systém

- Webové služby, ktoré používate už roky:
 - book-ordering services
 - search services
 - online dictionary services
- Všetko sú to RESTovské Webové služby
- Používali ste REST, stavali ste RESTovské webové služby a ani ste o tom nevedeli!
- **Google – koniec SOAP – už iba REST !**

REST: Príklad

- **Firma Sklad súčiastok s.r.o.**
- Sklad súčiastok – REST Web servis
- Umožniť zákazníkovi:
 - Získať výpis zoznamu súčiastok
 - Získať detailnú informáciu o konkrétnej súčiastke
 - Zaslať objednávku
- Ako bude služba implementovaná?

REST: Príklad



REST: Príklad

- Klient získa zoznam súčiastok na URL:
<http://sklad-suciastok.sk/suciastky>
- Klient obdrží dokument:

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.sklad-suciastok.sk"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part id="00345" xlink:href="http://sklad-suciastok.sk/suciastky/00345"/>
  <Part id="00346" xlink:href="http://sklad-suciastok.sk/suciastky/00346"/>
  <Part id="00347" xlink:href="http://sklad-suciastok.sk/suciastky/00347"/>
  <Part id="00348" xlink:href="http://sklad-suciastok.sk/suciastky/00348"/>
</p:Parts>
```

- Zoznam súčiastok má linky na zdroje detailných informácií o súčiastkach

REST: Príklad

- Služba má URL na zdroj informácií o súčiastke na linke:

<http://sklad-suciastok.sk/suciastky/00345>

- Klient obdrží dokument:

```
<?xml version="1.0"?>
<p:Part xmlns:p="http://sklad-suciastok.sk"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part-ID>00345</Part-ID>
  <Name>Widget-A</Name>
  <Description>This part is used within the frap assembly</Description>
  <Specification xlink:href="http://sklad-suciastok.sk/suciastky/00345/spec"/>
  <UnitCost currency="USD">0.10</UnitCost>
  <Quantity>10</Quantity>
</p:Part>
```

REST: Príklad

- Pre objednávku klient vytvorí objednávkový dokument podľa objednávacej schémy a odošle ho ako HTTP POST na url:
<http://www.sklad-suciastok.sk/objednavka/>
- Objednávková služba odpovie na HTTP POST s URL na odoslanú objednávku
<http://www.sklad-suciastok.sk/objednavka/1234>
- Teda klient môže kedykoľvek prístupit' k objednávke neskôr kvôli update/edit

REST: Zhrnutie

■ Výhody

- Linkovanie a bookmarkovanie – google friendly URLs
- Podpora štandardnej sady operácií
- Škálovateľnosť
- Jednoduchá implementácia – HTTP libs
- Slabá väzba komponentov (loose coupling)
- Možná neskorá väzba – HTTP status 302
- Vyššia možnosť znovupoužitia kódu

REST: Zhrnutie

■ Nevýhody

- Viazaný na HTTP
- Veľké množstvo objektov
- Správa URI menných priestorov (namespace) môže byť ťažkopádna – závisí na architektovi



Otázky



Vyberme si REST web framework

Peter Rybár

Web Framework

- Čo je to **Web Framework**?
 - Web framework je *softvérový framework* navrhnutý na podporu vývoja dynamických web stránok, Web aplikácii a/lebo Webových služieb
 - Cieľom je *uľahčiť prácu* spojenú s bežnými aktivitami pri webovom vývoji
 - Definuje **architektúru** aplikácie
 - MVC frameworky
 - Komponentovo orientované

Súčasný stav

- V súčasnosti existuje veľké množstvo javovských webových frameworkov
- Medzi najznámejšie môžeme zaradiť:
 - Struts (2)
 - Spring MVC
 - WebWork
 - Tapestry
 - JSF

Struts 2

■ Výhody

- považovaný za štandard
- jednoduchá architektúra
- komponentovo orientovaná alebo stránkovo orientovaná navigácia

■ Nevýhody

- slabá dokumentácia
- príliš sa zameriava na nové featury

Spring MVC

■ Výhody

- integrácia s veľkým počtom technológií na zobrazovanie: JSP/JSTL, Tiles, Velocity, FreeMarker, ODF, XSL, PDF
- ľahko testovateľný vďaka inverzii kontroly

■ Nevýhody

- náročná konfigurácia – veľa XML
- možno príliš flexibilný – žiaden spoločný rodičovský kontroler

WebWork

■ Výhody

- jednoduchá architektúra – ľahko rozšíriteľná
- Tag Library je ľahko upravovateľná pomocou FreeMarker alebo Velocity
- kontrolerovo alebo stránkovo orientovaná navigácia

■ Nevýhody

- malá komunita
- slabšia dokumentácia



Tapestry

■ Výhody

- produktívny, keď sa ho naučíte
- templaty sú HTML – dobre pre designerov
- početná užívateľská komunita

■ Nevýhody

- dokumentácia je viac konceptuálna ako pragmatická
- strmá krivka učenia sa
- spätná nekompatibilita

JSF

■ Výhody

- J2EE priemyselný štandard
- relatívne rýchly a jednoduchý vývoj
- bohatý navigačný framework

■ Nevýhody

- nerozumie si s **REST**om
- problémy s bezpečnosťou
- veľa XML konfigurácie
- dnes máme JavaEE namiesto J2EE

Súčasná situácia



Ake máme možnosti

- dozvedeli sme sa čo je to **REST**, jeho základné princípy
- prirodzená otázka teraz znie:
Aké možnosti vyvíjania web aplikácií RESTovským spôsobom máme s ohľadom na existujúce frameworky?

Prvý filter REST frameworkov

- niektoré vlastnosti existujúcich webových frameworkov sú nevyhovujúce pre vývoj **REST**ovských aplikácií
 - **stavový atribút** prenášaný POSTom/GETom
 - **URL** obsahujúca sessionId a pod. → stav nedefinuje **URL**
 - JSF (v súčasnej verzii), Tapestry,... – **nie sú vhodné** na vývoj **REST**ovských aplikácií

Čo naozaj potrebujeme

- framework na tvorbu webových aplikácií resp. webových služieb
- najlepšie ako **jeden** framework podporujúci vývoj webových aplikácií a webových služieb **REST**ovským spôsobom
- **jednoduchý** framework
 - bez tony XML konfigurácie
- spomínané Java web frameworky nespĺňajú úplne naše požiadavky :(

Architektúra frameworku

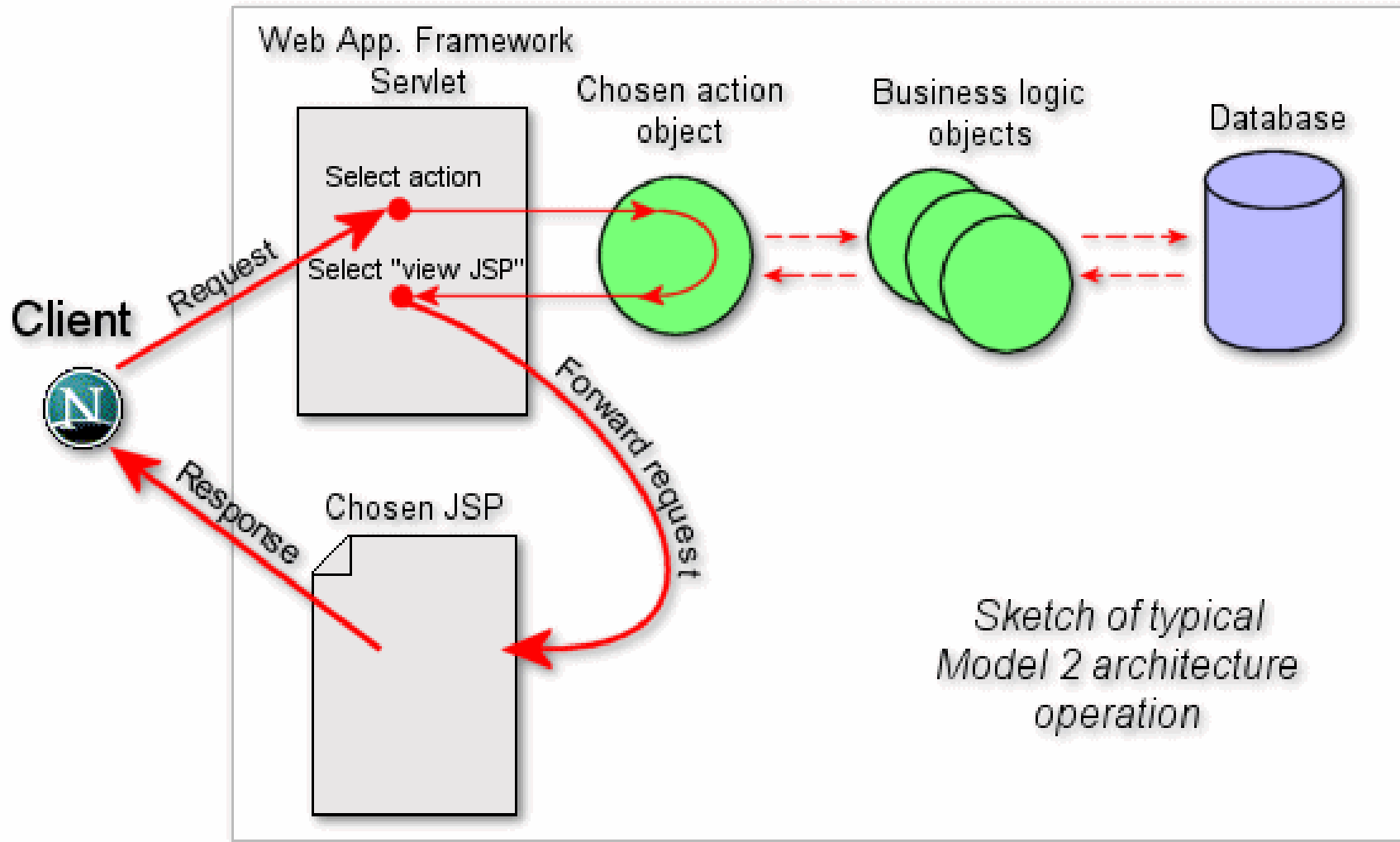
- webové aplikácie typicky obsahujú kód pracujúci s databázou, designom stránok, and control flow
- aby mohli byť väčšie aplikácie udržiavateľné, je potrebné mať tieto časti oddelené
 - jednou z možností na oddelenie týchto častí je použitie **MVC** architektúry
- príklady: Struts, Spring, WebWorks, ...

MVC

- rozdeľuje aplikáciu na 3 vrstvy
 - **model** – *biznis proces vrstva*
 - modeluje dáta a biznis procesy
 - **view** – *prezentačná vrstva*
 - zobrazuje výsledok biznis logiky (model)
 - **kontroler** – *riadiaca vrstva*
 - spojenie medzi užívateľskou interakciou a biznis procesmi v pozadí
 - zodpovedný za výber reprezentácie (view)



Server



Kontroler

- hlavným objektom nášho záujmu bude **kontroler**
- dva rôzne prístupy ku kontroleru
 - 1 kontroler – 1 metóda(command)
 - 1 kontroler – viacej metód

Ako by sme chceli vyvíjať RESToskú aplikáciu?

- nasledujúce ukážky sú v Jave, ale v princípe by tieto myšlienky mali byť implementovateľné v ľubovoľnej technológii/jazyku
- **kontroler** definujeme štandardným spôsobom dedením z rodičovského kontrolera

```
public class PartsDepotController extends WebController {  
  
}
```


Mapovanie kontrolera

- kontroler chceme mapovať na časť **URL** od rootu aplikácie

- `http://<server_url>/<app_root>/<controller_path>/`

- v aplikácii musíme definovať, kam namapovať kontroler

- dynamické mapovanie

- 1 kontroler - *viacero* URL

```
mount("/depot/", PartsDepotController.class);
```

- ako bude vyzerat' URL

- `http://company.net/depot/`

Mapovanie metód

- metóda je mapovaná na časť **URL** kontrolerom
 - `http://<server_url>/<app_root>/<controller_path>/method`
- existujú 2 prístupy k mapovaniu metódy
 - priame mapovanie mena metódy na URL
 - mapovanie mena metódy na symbolický názov, ktorý bude v URL

Vstupy z webu

- **vstup** môžeme z webu dostať **len v textovej podobe**
- **bežne** sme zvyknutí implementovať **metódy s parametrami ľubovoľných typov**
- potrebujeme podporu pre **konverziu základných typov** (čísla, dátum, a pod.)
- ako poslať vstupy
 - **URL parametre**: `<url>/param1/param2/.../paramN`
 - **GET**: `<url>?key1=value1&key2=value2`
 - **POST**: v HTTP body

Príklad

- <http://company.net/depot/parts>
- <http://company.net/depot/part/13>

```
public class PartsDepotController extends WebController {  
  
    public String parts() {  
        Parts parts = new BL().getParts();  
  
        String xmlOutput = Tools.toXML(parts);  
  
        return xmlOutput;  
    }  
  
    public String part(int id) {  
        Part part = new BL().getPart(id);  
  
        String xmlOutput = Tools.toXML(part);  
  
        return xmlOutput;  
    }  
}
```

Ako môže vyzerat' výstup

- <http://company.net/depot/parts>

- ```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://company.net"
 xmlns:xlink="http://www.w3.org/1999/xlink">
 <Part id="00345" xlink:href="http://company.net/depot/part/12"/>
 <Part id="00346" xlink:href="http://company.net/depot/part/13"/>
 <Part id="00347" xlink:href="http://company.net/depot/part/14"/>
</p:Parts>
```

- <http://company.net/depot/part/13>

- ```
<?xml version="1.0"?>
<p:Part xmlns:p="http://company.net"
        xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part-ID>13</Part-ID>
  <Name>Widget-A</Name>
  <Description>This part is used within the frap assembly</Description>
  <UnitCost currency="USD">0.10</UnitCost>
  <Quantity>10</Quantity>
</p:Part>
```

Mapovanie parametrov

- URL: <http://company.net/depot/part/13>

```
public String part(int id)
```

- GET: <http://company.net/depot/part?id=13>
 - keďže v Jave nie je možné pomocou reflexie zistiť meno parametru metódy, musíme mená mapovať nepriamo (napr. pomocou anotácií)
- POST: <http://company.net/depot/part>

```
public String part(@Key("id") int id)
```

Ďalšie možnosti mapovania

- **HttpServletRequest, HttpServletResponse**
 - ako by sme programovali metódu *servletu*
- všetky URL parametre
 - napr. ako typ **UrlParameters**
- všetky POST/GET parametre
 - napr. ako typ **RequestParameters, PostRequestParameters, GetRequestParameters**

Validácia vstupov

- veľmi dôležitá – bezpečnosť
- potrebujeme validovať dva druhy vstupov
 - jednotlivé vstupy (webové služby)
 - celý formulár (webové aplikácie)
- dve fázy validácie
 - **syntaktická** – prevod textového reťazca na požadovaný typ
 - **sémantická** - napr. kontrola veľkosti čísla a pod.

Výstupy na web

- výstup na webe môže byť len **text/stream**
- štandardne programujeme metódy, ktoré môžu mať ako návratový typ ľubovoľný objekt
- potrebujeme podporu pre automatickú **serializáciu** výstupu (napríklad do JSON)

```
@Json(maxdepth = 5)
public Part part(int id)
{
    Part part = new BL().getPart(id);
    return part;
}
```

Zobrazenie výstupu

- chceme v kóde lepšie **oddeliť logiku od prezentačnej vrstvy**

```
public void parts() {  
    Parts parts = new BL().getParts();  
    view("templates/parts.jsp", "parts", parts);  
}
```

```
@View(template = "templates/parts.jsp")  
public Part part(int id)  
{  
    Part part = new BL().getPart(id);  
    return part;  
}
```



Otázky



Webová aplikácia ako klient ku RESTovským webovým službám, AJAX

Peter Rybár



Klasické Web aplikácie

■ Výhody

- Jednoduchosť – dokumentovo orientované
- Dostupnosť – URL reprezentuje dokument
- Jednoduchá implementácia

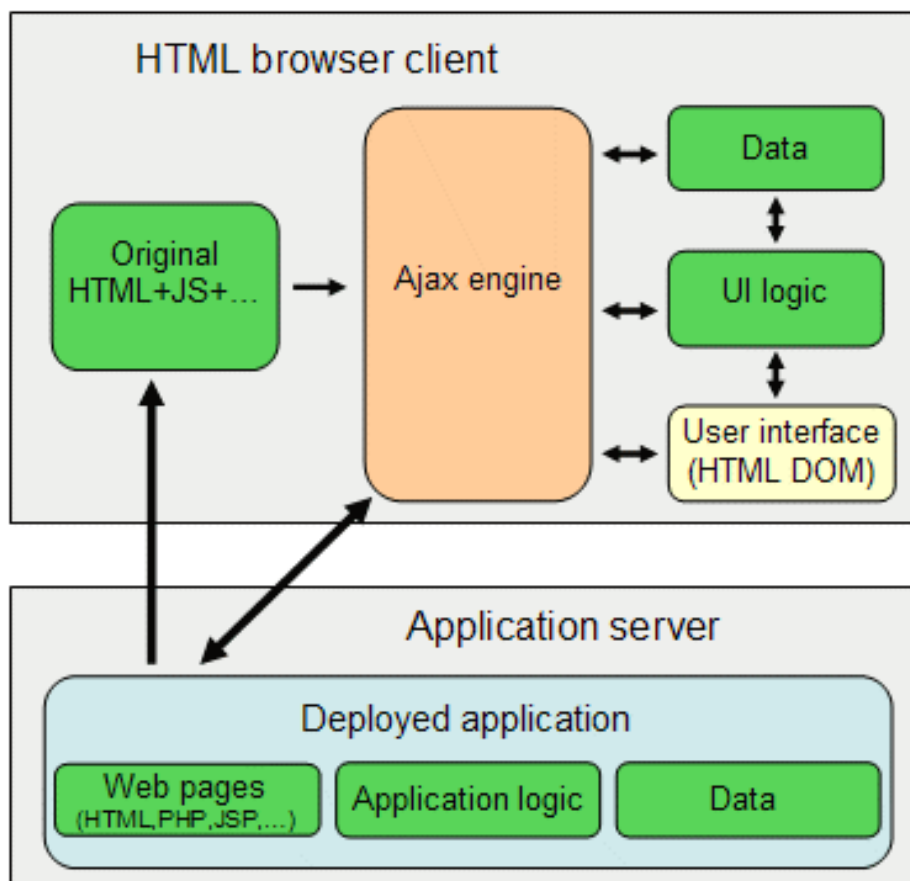
■ Nevýhody

- Nutnosť generovať kvôli malej zmene celú stránku
- Žiadna dynamika komponentov

Novodobé Web aplikácie

- **RIA** alebo **Web 2.0**
- **AJAX** - Asynchronous JavaScript and XML
 - Umožňuje asynchrónnu prácu s dátami
 - Nie je potrebné znovuzobrazenie stránky na zobrazenie nových dát
 - Jednoduché zavedenie dynamiky do webu
 - Umožňuje asynchrónnu prácu s dátami
- **Dátový zdroj** pre AJAX sú **RESTovské webové služby**

Novodobé Web aplikácie



- Možné rôzne formáty výmeny dát
 - XML
 - XHTML
 - JSON
 - YAML
- Dbáme na **sémantiku** dát

Formáty: XML (XHTML)

- XML – eXtensible Markup Language
 - <http://www.w3.org/XML/>
- Najznámejší jazyk na popis dát
- Ľahko párovateľný a rozšíriteľný
- Popis dát spolu s dátami
- REX: REST-Enabled XHTML
- Mikroformáty

Príklad - XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<breakfast_menu>
```

```
  <food>
```

```
    <name>Belgian Waffles</name>
```

```
    <price>$5.95</price>
```

```
    <description>two of our famous Belgian Waffles with plenty of real maple  
      syrup</description>
```

```
    <calories>650</calories>
```

```
  </food>
```

```
  <food>
```

```
    <name>French Toast</name>
```

```
    <price>$4.50</price>
```

```
    <description>thick slices made from our homemade sourdough bread</description>
```

```
    <calories>600</calories>
```

```
  </food>
```

```
</breakfast_menu>
```

Formáty: JSON

- **JSON** - JavaScript Object Notation
 - <http://json.org/>
- Ľahko čitateľný
- Vytvorený pre komunikáciu v jazyku JavaScript
- Podmnožina jazyka YAML
- Odporúča sa použiť pri komunikácii s webovým klientom – výkon a priame spracovanie do natívnych typov JavaScriptu

Príklad - JSON

```
{
  "menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        {"value": "New", "onclick": "CreateNewDoc()"},
        {"value": "Open", "onclick": "OpenDoc()"},
        {"value": "Close", "onclick": "CloseDoc()"}
      ]
    }
  },
  "id": "help",
  "value": "Help",
  "popup": {
    "menuitem": [
      {"value": "About", "onclick": "About()"}
    ]
  }
}
```

Formáty: YAML

- **YAML** – YAML Ain't Markup Language
 - <http://www.yaml.org/>
- Strojovo spracovateľný formát na serializáciu dát
- Ľahko čitateľný a pochopiteľný
- Vytvorený pre **skriptovacie jazyky**
- Umožňuje navyše ukazovatele (pointery)

Príklad - YAML

```
--- !clarkevans.com/^invoice
invoice: 34843
date : 2001-01-23
bill-to: &id001
  given : Chris
  family : Dumars
  address:
    lines: |
      458 Walkman Dr.
      Suite #292
    city : Royal Oak
    state : MI
    postal : 48046
ship-to: *id001
tax : 251.42
total: 4443.52
comments: >
  Late afternoon is best.
  Backup contact is Nancy
  Billsmer @ 338-4338.
```



Web 2.0 vlastnosti/techniky

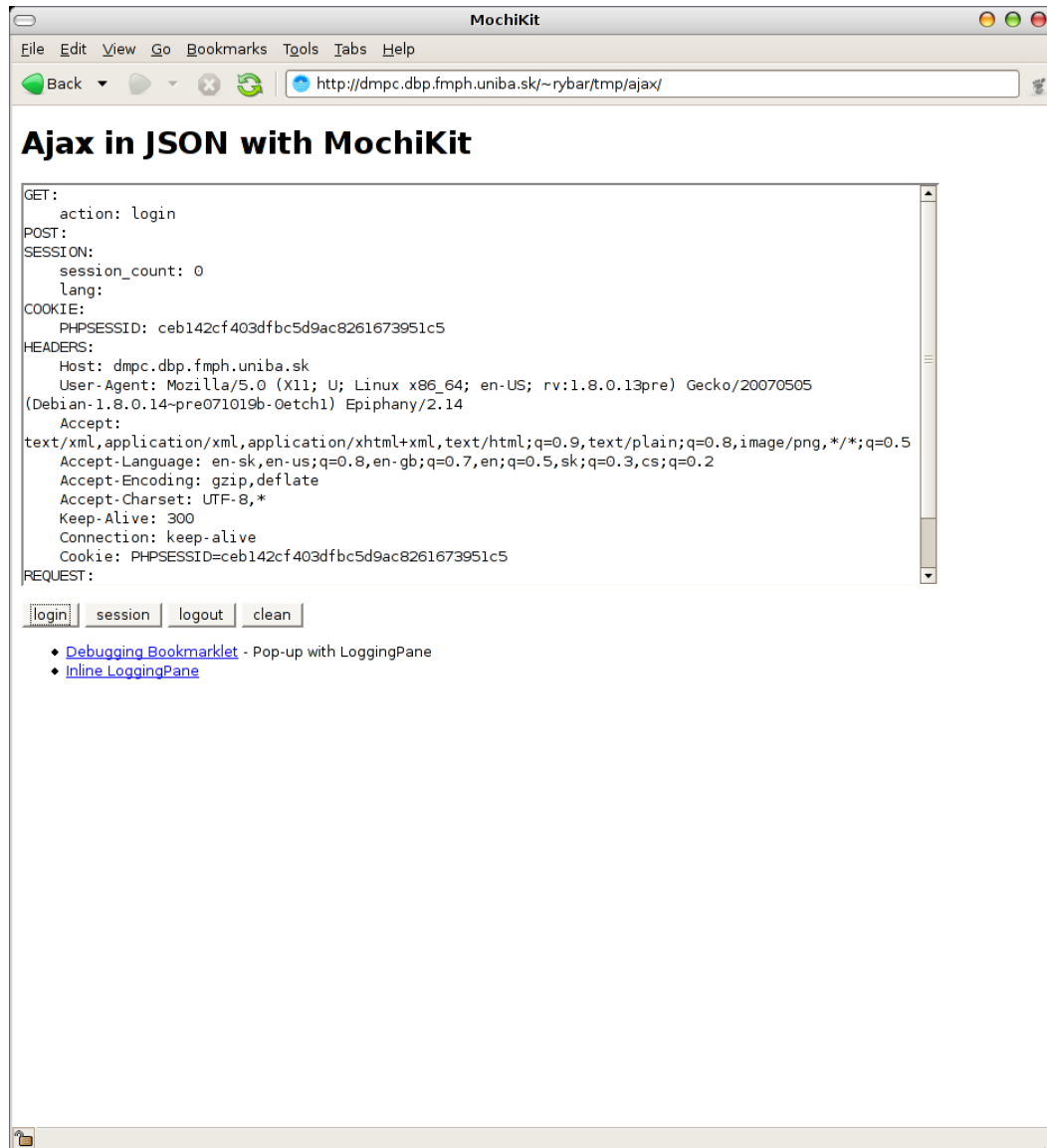
- **Rich Internet Application** techniky sú zvyčajne Ajax-based
- sémanticky valídne XHTML
- mikroformáty Rozširujú web stránky o novú prídavnú sémantiku
- **Cascading Style Sheets** separuje prezentáciu od obsahu



Web 2.0 vlastnosti/techniky

- REST a XML- a/alebo JSON-based API
- syndikácia, agregácia a notifikácia pomocou RSS alebo Atom feedov
- mashups, spájanie kontentu z rôznych zdrojov, klient- and server-side
- mashup ako nadmnožina starších portletov
- weblog-publishing tools
- wiki alebo fóra na podporu užívateľom generovaného obsahu

Weboví a neweboví klienti ku RESTovským web službám



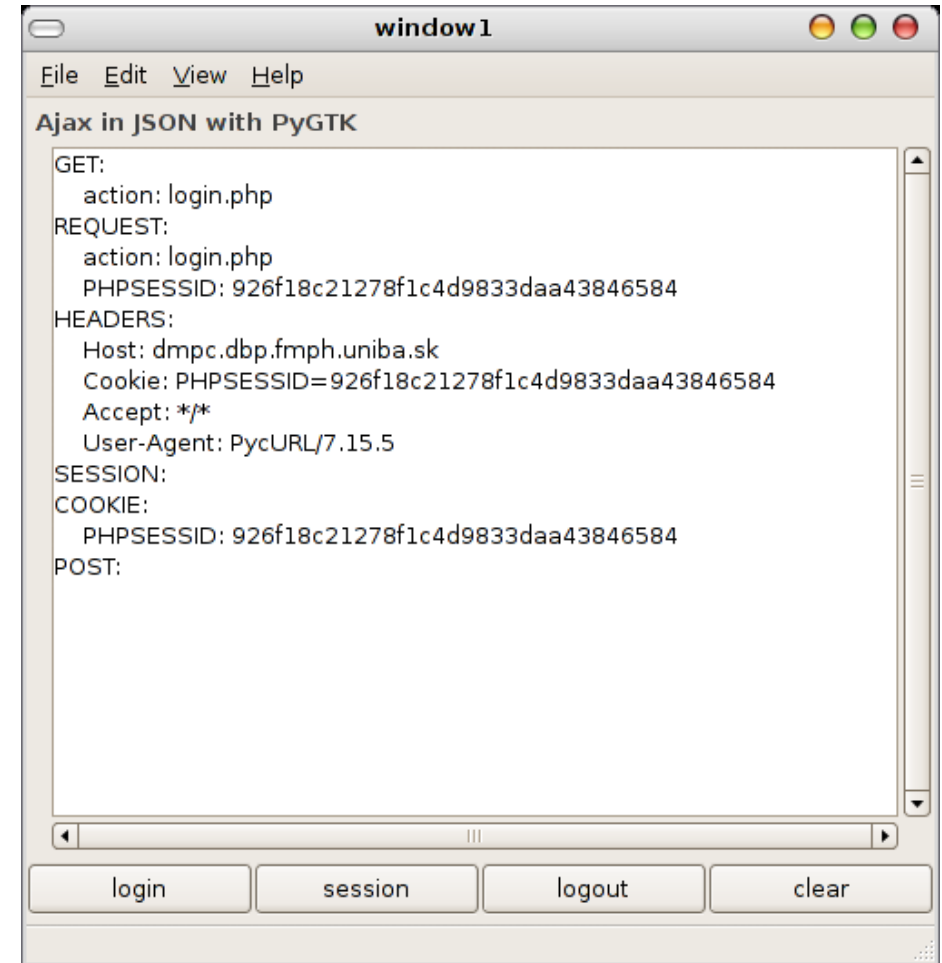
The screenshot shows the MochiKit web client interface. The browser window title is "MochiKit" and the address bar shows "http://dmpc.dbp.fmph.uniba.sk/~rybar/tmp/ajax/". The main content area displays the following request details:

```
GET:
  action: login
POST:
SESSION:
  session_count: 0
  lang:
COOKIE:
  PHPSESSID: ceb142cf403dfbc5d9ac8261673951c5
HEADERS:
  Host: dmpc.dbp.fmph.uniba.sk
  User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.0.13pre) Gecko/20070505
(Debian-1.8.0.14-pre071019b-0etch1) Epiphany/2.14
  Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
  Accept-Language: en-sk,en-us;q=0.8,en-gb;q=0.7,en;q=0.5,sk;q=0.3,cs;q=0.2
  Accept-Encoding: gzip,deflate
  Accept-Charset: UTF-8,*
  Keep-Alive: 300
  Connection: keep-alive
  Cookie: PHPSESSID=ceb142cf403dfbc5d9ac8261673951c5
REQUEST:
```

At the bottom, there are buttons for "login", "session", "logout", and "clean". Below the buttons, there are two links: "Debugging Bookmarklet - Pop-up with LoggingPane" and "Inline LoggingPane".

■ webový klient

■ newebový klient



The screenshot shows the PyGTK web client interface. The window title is "window 1" and the menu bar includes "File", "Edit", "View", and "Help". The main content area displays the following request details:

```
GET:
  action: login.php
REQUEST:
  action: login.php
  PHPSESSID: 926f18c21278f1c4d9833daa43846584
HEADERS:
  Host: dmpc.dbp.fmph.uniba.sk
  Cookie: PHPSESSID=926f18c21278f1c4d9833daa43846584
  Accept: */*
  User-Agent: PycURL/7.15.5
SESSION:
COOKIE:
  PHPSESSID: 926f18c21278f1c4d9833daa43846584
POST:
```

At the bottom, there are buttons for "login", "session", "logout", and "clear".

Newebový RESTovský klient

- Rozdiely od webového klienta:
 - Použitie iného programovacieho jazyka
 - Možnosť práce s dátami uloženými na lokálnom počítači
 - Väčšia diverzia v GUI
 - Možnosť vybrať si jazyk vyššej alebo nižšej úrovne
- Jazyky – Python, Java, Ruby, C++, ...

Newebový RESTovský klient

- Zhody s webovým klientom:
 - Rovnaký **protokol** na komunikáciu (HTTP) medzi klientom a serverom
 - Rovnaké webové služby na strane servera
 - **Jedna biznis logika** na serveri pre webových aj pre newebových klientov
 - Jednotné REST API pre klient-server komunikáciu
 - Rovnaký formát prijímaných a posielaných dát

Príklad - REST web aplikácií

- Webové aplikácie ponúkané firmami:
 - Yahoo!
 - <http://developer.yahoo.com/javascript/howto-ajax.html>
 - Google
 - <http://code.google.com/webtoolkit/>
 - Amazon
 - <http://developer.amazonwebservices.com/>



Zhrnutie

Zhrnutie

- **Architektúra** je dôležitá, **nie technológia**
- Pri tvorbe architektonického riešenia sa sústredíme na **protokoly, otvorené formáty a sémantiku dát**
- Výhodou je **nezávislosť riešenia na platforme**
- Dovoľme riešeniu rásť – **škálovateľnosť** a **modularita**
- Je dôležité **vedieť si správne vybrať**
- Pri návrhu riešení dodržiame „**Design Principles**“



Design Principles

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Errors should never pass silently. Unless explicitly silenced.



Design Principles

- Now is better than never. Although never is often better than right now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea – let's do more of those!



Otázky