

Gymnázium Dr. J. Pekaře Mladá Boleslav

Maturitní práce



DIGITALIZACE FORMULÁŘE

pomocí programovacího jazyka PHP

Předmět: Informatika a výpočetní technika

Datum a rok odevzdání:

29.3.2014

Jméno a příjmení:

Pavel DVOŘÁK

Ročník: **8.O**

Počet stran: **33**

Počet příloh: **1**



Prohlášení

Prohlašuji, že jsem tuto maturitní práci vypracoval samostatně a že jsem uvedl všechny literární i elektronické prameny, z nichž jsem čerpal. Dále prohlašuji, že jsem při práci na projektu, jež je součástí této maturitní práce, nijak nezneužil autorská práva, a že jsem všechny licence k použitému kódu nebo softwaru respektoval a případně je řádně uvedl.

.....

Obsah

1	Úvod	4
2	Představení použitých technologií	7
2.1	HTML	7
2.2	CSS	7
2.3	JavaScript.....	8
2.4	SQL a MySQL	9
2.5	PHP	9
2.6	Server	9
2.7	Editor.....	10
2.8	Písmo.....	10
2.9	Ikonové písmo.....	10
3	Uživatelská část.....	12
3.1	Základní struktura formuláře	12
3.2	Hlavička	13
3.3	Tělo	14
3.3.1	Struktura stránky	14
3.3.2	Formulářová pole	14
3.3.3	Navigace.....	16
3.4	Vzhled	16
3.4.1	Stránka.....	16
3.4.2	Formulářové prvky	18

3.4.3	Navigace.....	20
3.4.4	Písmo.....	20
3.4.5	Prefixy	21
4	Strojová část.....	23
4.1	PHP v HTML kódu.....	23
4.2	Úvod do sessions.....	23
4.3	Limitace sessions	24
4.4	Zápis dat do cookies.....	25
4.5	Zápis dat do databáze.....	26
4.6	Zápis dat do souboru.....	28
4.7	Odeslání potvrzujícího emailu	29
4.8	Získání zapsaných dat.....	30
4.9	Vícenásobné vkládání kódu	32
4.10	Detekce a oprava chyb	33
5	Závěr.....	36
	Seznam zdrojů.....	37
	Seznam obrázků	38
	Seznam ukázek kódu.....	39
	Seznam příloh.....	41

1 Úvod

V dobách, kdy technologie propojení počítačů pomocí počítačových sítí za účelem sdílení souborů, dat a webových prezentací pozvolna zaplavuje lidstvo, většina lidí používajících počítač a internet hledá způsob, jak by si díky novým technologiím zjednodušila práci.

Právě v těchto dobách také vychází z módy klasické papírové formuláře a pozvolna je nahrazují ty internetové. Jsou přehledné a jednodušeji uspořádané, uživatel je okamžitě upozorněn, jestliže udělá chybu a může ji snadno ihned opravit, tyto formuláře jsou také lépe strojově i klasicky čitelné, díky čemuž jsou moderní a široce oblíbené.

Jenže i přes to popularita programovacích jazyků příliš neroste. Naprostá většina lidí stále nemá s programováním mnoho zkušeností, a proto každý, kdo potřebuje z jakéhokoli důvodu vytvořit jednoduchý webový formulář, řeší vzniklý problém velmi těžce. Pokud opravdu chceme jít s dobou a využít výhod webového formuláře, pak se nám nabízí více možností řešení takového problému. První možností je vytvořit daný formulář pomocí kancelářského balíku. Je to možnost sice nejjednodušší, ale postrádá mnoho výhod těch klasických webových formulářů, jaké dobře známe. Druhou možností je najmout si někoho, kdo takový webový formulář vytvoří za nás. Ale podmínkou je, mít dostatek financí, což v dnešní „šetřivé“ době není zcela ekonomicky výhodné.

A tak některým nezbyvá, než nějaký jednoduchý internetový formulář vypracovat vlastními silami. V tuto chvíli se obvykle naprostý nováček musí „prokousat“ skrz základy programování vůbec, musí se poprat hned s minimálně čtyřmi programovacími jazyky najednou a často musí také dešifrovat kód po svém předchůdci. Zde už závisí právě na člověku samotném, jakým způsobem chce nebo spíše může daný problém řešit.

První možností tvorby vlastního webového formuláře je tzv. redakční systém (například Drupal nebo Wordpress). V tomto případě je proniknutí do takového systému často i bez jakýchkoli znalostí programovacích jazyků jednoduché a intuitivní. Tento člověk má štěstí a může se směle pustit do tvorby, přičemž má k dispozici nespočet knih, manuálů, dokumentace i návodů od fanoušků a to nejen na internetu, ale i v knihkupectví nebo živě na přednáškách a diskusích. Jenže tahle skvělá možnost má svůj háček. Často by

to totiž znamenalo přetvořit celý původní, již dříve vytvořený web, do čehož se pouštět je nejen časově, ale i fyzicky a psychicky náročné.

Proto se člověk často rozhodne napsat si takový formulář sám a ručně, přičemž na internetu příliš mnoho „kuchařek“ k tomuto tématu nenajde, u knih takový člověk netuší, jakou příručku má zakoupit, a na přednáškách se už bohužel řeší pouze poněkud pokročilé funkce, kterým z pozice začátečníka rozumět nelze. Na internetových fórech se tedy často setkáme s otázkami typu, co je tenhle (často velice triviální) úsek kódu zač a co dělá?

Rozhodl jsem se tedy takovouto kuchařku vytvořit. Cílem mojí práce je naučit kohokoliv s mírnými základy HTML a programování webových prezentací, jakým způsobem vytvořit jednoduchý, přehledný, ale hlavně funkční webový formulář a jakým způsobem jej umístit na web. Pro názornost jsem si vybral jeden konkrétní, se kterým jsem se v závěrečném ročníku střední školy setkával v digitální i papírové podobě asi nejvíce. Jedná se o přihlášku ke studiu na vysoké škole.

Pro nastínění řešení toho problému jsem si vybral programovací jazyk PHP. Jedná se o „skriptovací programovací jazyk, který je určený především pro programování dynamických internetových stránek a webových aplikací například ve formátu HTML“ (Příspěvatelé Wikipedie, 2014). Jeho nejčastějším využitím je zajištění komunikace mezi klientem a serverem, což je pro zpracování dat z formulářů velice důležitá vlastnost. Mezi jeho největší přednosti dále patří přehledný manuál, a dokumentace k dispozici online, jeho dynamický vývoj, který zachovává ve většině případů zpětnou kompatibilitu¹, a naprostá jednoduchost i pro úplné začátečníky. Díky tomu a nejen díky tomu je velmi oblíbený a rozšířený napříč webhostingy.

V práci tedy nejprve čtenáře seznámím s jednotlivými technologiemi a dalšími programovacími jazyky, které pro vytvoření formuláře budou potřeba. Následně se

¹ Zpětná kompatibilita je terminus technicus pro stav, kdy je daný technický produkt (například kód nebo software) technicky plnohodnotně zaměnitelný se svojí starší verzí. V programátorské praxi se tak jedná o stav, kdy napsaný kód nezestárne a není potřeba jej po určité době aktualizovat, doplnit nebo úplně vyměnit (Příspěvatelé Wikipedie, 2013).

zaměřím na uživatelskou část. Rozvrhnu základní strukturu pomocí programovacího jazyka HTML a představím základní principy pro vzhled a uživatelskou přívětivost formuláře. Poté přejdu k té složitější, avšak neméně důležité strojové části. Zaměřím se na vygenerování formuláře počítačem, zpracování údajů do něj vložených a jejich zápis do šablony, odeslání na server nebo emailem, či uložení v klientské části. A na závěr představím další techniky uložení výstupu nebo jeho zpracování, popřípadě vyvolání výstupu kvůli jeho dalšímu použití, zreviduji a otestuji vytvořený formulář a případně dovysvětlím některé pojmy nebo drobné problémy, které by mohly při návrhu nastat.

2 Představení použitých technologií

Jak jsem již naznačil v úvodní kapitole, webové prezentace nesestávají z jednoho programovacího jazyka, který by zajišťoval veškeré interakce mezi uživatelem, klientem a serverem. K různým záměrům používáme různé programovací jazyky a v našem případě tomu není jinak.

2.1 HTML

První a nejzákladnější jazyk, který budeme potřebovat, je jazyk HTML (zkr. HyperText Markup Language, česky Hypertextový značkový jazyk). Ten se stará o základní rozvržení webové stránky a označení jednotlivých prvků v ní. Tímto jazykem vytvoříme strukturu všech stránek a částí formuláře, vložíme do ní jednotlivé formulářové prvky a také vytvoříme šablonu samotné přihlášky, do které se budou zadaná data doplňovat. Budeme také potřebovat použít některé moderní technologie. Ty však nejsou úplně zpětně kompatibilní, proto si vypomůžeme tzv. polyfily², které nám jejich funkčnost zajistí.

2.2 CSS

Druhou základní složkou, doplňující HTML je využití programovacího jazyka CSS (zkr. Cascading Style Sheet, česky Kaskádové styly). Tento jazyk určuje vzhled samotných HTML prvků. Většina klientů má již pro samotné prvky přednastavený styl vzhledu, některé styly jsou dokonce zakomponovány do samotné specifikace HTML. Každopádně takový základní vzhled rozhodně nestačí k vytvoření uživatelsky přehledného a příjemného formuláře, pro naše účely se tedy budou některé úpravy hodit. Výhodou Kaskádových stylů je možnost ke každému prvku HTML struktury přiřadit libovolný počet libovolných možností nastavení vzhledu jednoduchou formou. V poslední době se snaží klasické CSS

² Polyfill je úsek kódu nebo samostatná knihovna vložená do stránky z důvodu zajištění funkčnosti dané funkce napříč klienty. Používá se často k záplatování chyb, doplnění nadstandardních funkcí nebo kvůli zpětné kompatibilitě. Mnoho polyfillů je však naprogramováno v JavaScriptu, proto na ně nelze příliš spoléhat (viz kapitola 2.3 JavaScript) (Příspěvatelé Wikipedie, 2014).

nahradit tzv. CSS preprocesory, které rozšiřují samotné CSS o ještě rozsáhlejší kaskádovost nebo o funkce skriptovacích jazyků (například o proměnné). Jsou však o mnoho složitější a jejich pochopení vyžaduje pokročilé znalosti programování, proto pro naše potřeby bude bohatě stačit základní forma CSS.

2.3 JavaScript

Jedním z dalších základních programovacích jazyků pro návrh webu je široce rozšířený JavaScript (čti „džavaskript“). Jedná se o skriptovací jazyk, jehož základní předností ve tvorbě webových stránek je jeho výrazná podpora práce s DOM³ stránky, díky čemuž je nejvíce rozšířen mezi webovými prohlížeči. Jeho nevýhody však velmi markantně ovlivňují tvorbu webových aplikací, proto není příliš vhodné na něm stavět funkčnost celé stránky a proto jej také používat pro tento formulář až na výjimečné případy nebudeme. Pro názorný příklad jsem vybral ty nejmarkantnější nevýhody, jakými JavaScript trpí:

- 1) Uživatel si jej může ve svém klientovi (v našem případě ve webovém prohlížeči) úplně vypnout
- 2) Prohlížeče (převážně mobilní) často vypnutí JavaScriptu provádějí také automaticky, pokud je pomalé internetové připojení nebo pokud vědí, že dané zařízení (například mobilní telefon) nebude schopno tento kód správně interpretovat
- 3) Podpora jeho jednotlivých funkcí napříč prohlížeči se velmi liší a ani v nejnovějších prohlížečích nefungují funkce z několika let starých specifikací. Dalo by se říci, že webové prohlížeče si JavaScript interpretují každý po svém.
- 4) JavaScript je díky podmínkám jeho vzniku (vznikl v době označované jako „válka prohlížečů“ a byl jejím hlavním předmětem sporů, každý prohlížeč totiž vyvíjel vlastní skriptovací jazyk pro tyto účely a JavaScript byl ten,

³ DOM (zkr. Document Object Model, česky Objektový model dokumentu) je rozhraní umožňující skriptovacím jazykům přístup ke struktuře stránky, k jejím jednotlivým prvkům a případně také k jejich důležitým vlastnostem (Příspěvatelé Wikipedie, 2014).

který zvítězil) velice programátorsky neintuitivní a obtížně použitelný s mnoha chybami a nejasnými funkcemi

- 5) Jeho funkce jsou také velmi zastaralé nebo zastarale interpretované, mluví se o jeho nahrazení v dohledné době, to však pomalu způsobuje možné další pokračování války prohlížečů, protože každý prohlížeč se připravuje na jeho nahrazení jiným jazykem (často jazykem z vlastních dílen).

2.4 SQL a MySQL

Další důležitý jazyk pro naplnění našich představ a potřeb pro zpracování zadaných dat z formuláře je jazyk SQL (zkr. Structured Query Language, česky Strukturovaný dotazovací jazyk). Jako takový je vhodný pro mnoho účelů, často při komunikaci mezi dvěma zařízeními. Jeho nejčastější využití však spočívá v ukládání dat do databází a jejich vyvolávání zpět. Na jazyku SQL je postaveno mnoho databázových systémů. Velmi rozšířenou mezi webhostingy a také velmi populární mezi programátory je databáze MySQL (Zajíc, 2005). Databázový systém MySQL je navíc velice rychlý a spolehlivý, proto jej využijeme ke správě naší databáze vyplněných a odeslaných formulářů. Je nezbytně nutné, aby bylo rozhraní MySQL na serveru nainstalováno jakožto CGI skript (jakožto rozšíření Apache serveru), nikoliv jako spustitelný program a to alespoň ve verzi 5, nejlépe však v té nejaktuálnější.

2.5 PHP

A konečně se na závěr dostáváme k jádru celého našeho počínání. PHP (rekurzivní zkr. PHP: Hypertext Preprocessor, česky PHP: hypertextový preprocesor). Ten jsem již představil v úvodu, proto teď nebudu zabíhat do podrobností. Je nezbytně nutné, aby bylo rozhraní PHP na serveru nainstalováno také jakožto CGI skript a také alespoň ve verzi 5, nejlépe však v té nejaktuálnější. Podpora jednotlivých funkcí u různých verzí PHP se však mírně liší, proto raději upozorním, pokud použiji funkci, která vyžaduje ještě novější verzi, než je verze 5.

2.6 Server

Pro naše účely bude důležité, aby webové stránky běžely na tzv. Apache HTTP serveru. Podle Wikipedie v současné době dodává prohlížečům na celém světě většinu

internetových stránek (Příspěvatelé Wikipedie, 2014), proto si myslím, že by s jeho zajištěním na straně webhostingu neměl být problém. Budeme si na něm ukazovat některá drobná nastavení. Samozřejmě ta jdou provést také v dalších serverech (jako např. Nginx), ovšem vyžaduje to pokročilou orientaci v takovém serveru a převedení všech příkazů a nastavení z Apache, která si budeme ukazovat.

2.7 Editor

Jednou z důležitých pomůcek je bezesporu také kvalitní textový editor. Na výběr je mnoho možností od těch nejzákladnějších, jako například poznámkový blok nebo tzv. Notepad2, až po ty pokročilejší, mezi kterými bych vybral například český PSPad. O výběr rozhodně není nouze, je však důležité, aby byl programátor na editor zvyklý a aby se mu v něm dobře pracovalo. Pro jednoduchou a přehlednou správu projektů je doporučován i editor Brackets, který použiji k ukázkám kódu.

2.8 Písmo

Najít vhodné písmo pro webový formulář nebo čistě pro webovou stránku je velmi složitý oříšek. Naštěstí je pro tyto potřeby na internetu k nalezení mnoho zajímavých článků o „typeface“, čili o využití různých druhů písem při různých příležitostech. Těmi se však mnoho zabývat nebudeme, protože to není předmětem této práce. Nejjednodušší pro nás bude využít písmo z databáze Google Fonts, která nabízí možnost vybrat si hezké a designově propracované písmo a vložit jej přímo do webové stránky. Doporučuji ještě využít aplikaci českého vývojáře zvanou „Fontokuk“ (Michálek, 2014), která z písem dostupných v databázi Google Fonts vybere pouze ty, které zobrazují českou diakritiku. Pro naše účely jsem jako příklad vybral uživatelsky příjemné a velice oblíbené a rozšířené písmo Ubuntu.

2.9 Ikonové písmo

Poslední důležitou roli v navigaci ve formuláři hrají moderní ikonová písma. Jde o klasická písma pro zobrazení textu, namísto písmen textu se však zobrazí ikonky. Písma se dají buď stahovat a používat jednotlivě jako celek (tak jako klasická písma) nebo si lze pomocí různých online služeb namíkovat z dostupných ikoněk písmo dle vlastního výběru.

My to však nebudeme komplikovat a pro názorný příklad využijeme jedno z již přednastavených písem MFG Labs icon set, které je dostupné ke stažení zdarma.

3 Uživatelská část

3.1 Základní struktura formuláře

I pro nováčky v HTML by tato kapitola měla být spíš jednodušší. Začneme nejprve základní strukturou webové stránky.

```
1 <!Doctype Html>
2 <Html>
3 <Head>
4 <Meta Charset="Utf-8">
5 <Title></Title>
6 </Head>
7 <Body>
8 </Body>
9 </Html>
```

Ukázka 1: Základní struktura HTML stránky

Každá HTML stránka by měla obsahovat nejprve typ dokumentu, neboli tzv. *doctype*. V něm je deklarován formát dokumentu. Samotný obsah stránky ohraničují párové značky `<html>`. Obsah se dělí na hlavičku (párové značky `<head>`) a tělo (párové značky `<body>`). Do hlavičky se zapisují informace, které nejsou uživateli viditelné a slouží pro zpracování strojově. Naopak do těla se píše přímo samostatný obsah stránky. Do hlavičky ještě doplníme titulek stránky (párové značky `<title>`), který se zobrazí v názvu otevřené záložky v prohlížeči.

Nesmíme ještě zapomenout na nepárovou značku `<meta>` s vlastností *charset*, čili sadou znaků. Základní sada znaků je v českém prostředí většinou *windows-1250*, pokročilá zase třeba *iso-8859-2*, ale nejuniverzálnější pro náš účel bude mezinárodní sada znaků Unicode (*utf-8*). Každý editor by měl nějakým způsobem nabídnout alespoň dvě sady znaků k výběru ať už při psaní, nebo při ukládání dokumentu. Sadu znaků později už měnit nemůžeme, proto si ji nadefinujeme teď. Zvolenou sadu znaků musíme poté použít naprosto stejnou všude, kde bude potřeba.

Takto připravenou šablonu zapíšeme tedy do editoru a uložíme s příponou *.php*. Úvodní stránka by se na všech serverech Apache měla jmenovat *index.xyz*, proto náš první soubor bude mít výsledný název *index.php*. Pokud se náš formulář nevejde na jednu stránku, další stránky můžeme buď sloučit do jednoho dokumentu a pomocí skriptovacích jazyků upravit přecházení mezi nimi, nebo mnohem jednodušeji rozdělit formulář do více

stránek. Pro každou další stránku poté můžeme zvolit název, jaký se nám zlíbí. Avšak opatrně, nejlépe bez diakritiky a samozřejmě tak, aby se název vztahoval k obsahu.

Ještě podotknu, že vše, co jsme zatím do souboru napsali, může být tzv. „case insensitive“, čili může být zapsáno jak velkými, tak malými písmeny, a může být zapsáno také kombinovaně. V ukázkách používám první písmena velká pouze pro přehlednost.

3.2 Hlavička

Do hlavičky bychom měli zapsat ještě další informace o dokumentu. Jednak by měla obsahovat zadání písem a stylů, také bychom měli uvést tzv. SEO údaje o stránce pro webové vyhledávače, případně deklarovat další aplikace a polyfilly, které použijeme.

```
1 <Meta Charset="Utf-8">
2 <Meta Name="Application-name" Content="">
3 <Meta Name="Author" Content="">
4 <Meta Name="Description" Content="">
5 <Meta Name="Generator" Content="">
6 <Meta Name="Keywords" Content="ukázka,práce,maturita">
7 <Meta Name="Robots" Content="Follow,index">
8 <Link Type="Image/x-icon" Rel="Shortcut icon" Href="favicon.ico">
9 <Link Type="Text/css" Rel="Stylesheet" Href="aplikace/pismo.css">
10 <Link Type="Text/css" Rel="Stylesheet" Href="aplikace/ikonkovepismo.css">
11 <Link Type="Text/css" Rel="Stylesheet" Href="styly.css">
12 <Style Type="Text/css">
13 // Další styly
14 </Style>
15 <Title></Title>
```

Ukázka 2: Hlavička stránky

Dalšími `<meta>` značkami a jejich vlastnostmi (viz Ukázka 2) doplníme tedy libovolné popisky pro webové vyhledávače. K dispozici máme název, autora, popis, generátor (jinými slovy editor) a klíčová slova (oddělují se čárkou). Další `<meta>` značka s vlastností *robots* udává, jakým způsobem se mají vyhledávače k obsahu chovat. *follow* znamená, že mají následovat odkazy a ukládat také stránky, na které odkazujeme. *index* zase určuje, že se ukládá stránka samotná, a *archive* určuje, zda se má stránka archivovat (ukládat její kopie vyhledávači, či archivátoru stránek). Případnou negaci provedeme přidáním předpony *no* (př.: *nofollow*).

Následují nepárové značky `<link>` pro vkládání obsahu do stránky. První vloží ikonu stránky, která se zobrazí u jejího titulku. Ikona by se měla kvůli zpětné kompatibilitě jmenovat *favicon.ico* a měla by se nacházet v kořenovém adresáři naší webové stránky (tzv. *rootu*). Další jsou písma, jejichž vkládání je popsáno v dokumentaci u každého písma,

případně písmo vložíme přímo do našich stylů (viz kapitola 3.4 Vzhled). Na závěr vložíme naše styly buď v externím souboru, nebo přímo.

3.3 Tělo

3.3.1 Struktura stránky

Základní rozvržení stránky samotné pomocí HTML bude v našem případě sestávat ze dvou navigačních menu a samotného formuláře.

```
1 <Div Class="Menu">
2 </Div>
3 <Form Action="zpracovani.php" Id="Formular" Method="Post">
4 </Form>
5 <Div Class="Menu">
6 </Div>
```

Ukázka 3: Tělo stránky

Navigační menu jsem ohraničil párovými značkami `<div>` používanými k ohraničení více prvků a označil třídou, abychom na ně mohli později aplikovat různé styly vzhledu. K formuláři (párové značky `<form>`) patří vlastnosti *action* a *method*. Vlastnost *action* určí soubor nebo skript, který zpracuje zadaná data formuláře. Vlastnost *method* zase určuje způsob předávání zadaných dat ke zpracování. Metoda *GET* předává data přímo v adrese url, ta jsou sice jednodušejší zpracovatelná různými druhy jazyků, ale kvůli nízké bezpečnosti způsobené transparentností těchto dat se tuto metodu nedoporučuje používat. Oproti tomu metoda *POST* předává data skrytě, na pozadí. Taková data jsou sice čitelná užším výběrem jazyků, lépe se však hodí pro větší objem dat a zaručují vyšší bezpečnost, proto jsem se rozhodl metodu *POST* použít. Na závěr do formuláře dopíšeme ještě vlastnost *id*, kterou formulář pojmenujeme pro případ, kdy budeme chtít některý z prvků formuláře (například odesílací tlačítko) umístit mimo samotný formulář (například do jednoho z menu – viz kapitola 3.3.3 Navigace).

3.3.2 Formulářová pole

Nejdůležitější součástí formuláře jsou pro nás jednotlivá formulářová pole. Vkládají se pomocí prvků `<input>`, `<textarea>`, `<select>`, `<option>` a `<button>`. Více prvků můžeme rozdělit do skupin párovými značkami `<fieldset>`, do kterých se nepovinně uvádí také párové značky `<legend>`, jejíž úlohou je nadřazené skupině přidělit název. Obecně se

doporučuje značku *fieldset* používat, jelikož tato značka fakticky zpřehlední a zjednoduší orientaci uživatele ve formuláři.

```
1 <Fieldset>
2 <Legend></Legend>
3 <Label For=""></Label>
4 <Input Id="" Name="" Tabindex="1" Type="Text" Value="">
5 <Textarea Name="" Tabindex="2"></Textarea>
6 <Select Name="" Tabindex="3">
7 <Option Value=""></Option>
8 </Select>
9 <Button Name="" Tabindex="4" Type="Submit" Value="">Odeslat</Button>
10 </Fieldset>
```

Ukázka 4: Jednotlivé prvky formuláře

Do jednotlivých skupin poté vkládáme prvky uvedené výše. Nepárovými značkami *<input>* vkládáme samotná vstupní formulářová pole. Vlastností *type* poté určíme, jakého typu bude toto pole. Na výběr máme například nejzákladnější textové, zaškrťovací, prepínací a číselné pole, dále pole pro vkládání data, času, barvy, telefonu nebo emailu. Jenom upozorním, že pole pro vkládání data, času a barvy nejsou příliš bezpečná k používání, protože je starší prohlížeče nepodporují. Pro náš záměr však potřeba nebudou, proto se nemusíme příliš bát. Jejich vložení by kvůli zpětné kompatibilitě vyžadovalo doplnění o nějaký JavaScriptový polyfill. Naštěstí je ve většině případů můžeme nahradit obyčejným textovým polem.

Párovými značkami *<textarea>* můžeme vložit pole ke psaní delších textů. Párové značky *<select>* zase vkládají pole pro výběr hodnoty ze seznamu. Jednotlivé hodnoty se vkládají pomocí párových značek *<option>*. Posledním často používaným prvkem HTML formulářů je tlačítko. Zapisuje se párovými značkami *<button>* a vlastnost *type* určuje, jakým způsobem se bude tlačítko chovat. Parametr *reset* vynuluje veškerá zadaná data, *submit* data odešle a *button* plní funkci bezvýznamového tlačítka, kterému můžeme přiřadit funkci skriptovacími jazyky. Text, který chceme v tlačítku zobrazit, pak můžeme vložit mezi párové značky.

Všechny formulářové prvky dále obsahují dvojici vlastností *name* a *value* (viz Ukázka 4), které určují název a přednastavenou hodnotu předávaných dat. Pro jednodušší orientaci handicapovaných uživatelů ve formuláři je také důležité nastavit vlastnost *tabindex*, která určuje pořadí prvků při přepínání mezi nimi pomocí klávesy TAB (například u nevidomých je tato vlastnost klíčová).

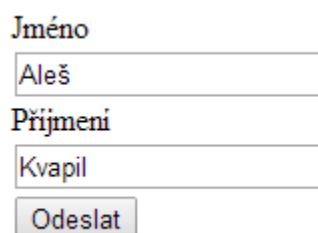
Do formulářových polí můžeme ale také vkládat vlastnosti, které nemají žádnou hodnotu a plní funkci parametru. Mezi těmito vlastnostmi formulářových prvků bych vyzdvihl *required*, jehož přítomnost zamezí odeslání formuláře ke zpracování, pokud je vybrané pole prázdné, nebo třeba *autofocus*, jehož přítomnost zajistí automatické přesunutí kurzoru do vybraného pole při načtení stránky. Ke každému poli se ještě přidává párovými značkami označený prvek `<label>` sloužící jako jeho popis. Používá se s vlastností *for*, do které se vepíše id formulářového pole, ke kterému se prvek váže.

3.3.3 Navigace

Součástí kvalitního formuláře by měla být také kvalitní navigace. Navigací myslím stránkování a intuitivní přecházení mezi jednotlivými stránkami, možnosti uložení anebo odeslání formuláře v průběhu i na konci vyplňování. Velkou roli v tomto ohledu hrají právě tlačítka a skupiny polí, která jsme si připomenuli v předchozí kapitole. Veškerou práci v našem formuláři přenecháme souboru *zpracovani.php*, proto můžeme u všech tlačítek nastavit vlastnost *type* na hodnotu *submit*. Pro tlačítka v navigačních menu (tj. mimo formulář samotný) musíme ještě dodat vlastnost *form*, jejíž hodnotu nastavíme na id formuláře, ke kterému se tlačítko váže.

3.4 Vzhled

Chceme-li, aby byl formulář „user-friendly“, čili uživatelsky příjemný k vyplnění, nesmíme jej zanechat v přednastavených stylech. Pole by v takovém formuláři neměla správnou délku pro určený obsah a celý formulář by poté působil stísněně a nepřehledně (Ott, 2012).



Jméno
Aleš

Příjmení
Kvapil

Odeslat

Obrázek 1: Jednoduchý formulář bez nastavených stylů

3.4.1 Stránka

Pro základní rozestavení a barevné schéma stránky budeme nyní pracovat s předpřipraveným HTML dokumentem. Znovu musím dodat, že u všech vlastností, pokud neuvedu jinak, nezáleží na velikosti písmen, pouze já jsem použil první písmena velká pro přehlednost.

Kaskádové styly sestávají vždy z označení HTML prvku, za nímž následují složené závorky. Do těch se poté vkládají jednotlivé styly oddělené středníkem. Každý jednotlivý styl je označen názvem, následuje dvojtečka a za ní hodnota. Styly vkládáme buď do HTML stránky přímo (do hlavičky mezi párové značky `<style>`) nebo do odděleného souboru s příponou `.css`, který do HTML stránky vložíme nepárovou značkou `<link>` (viz kapitola 3.2 Hlavička).

```
1 Body{Background-color:#FFBF00;Color:#242424;Font-family:Ubuntu,"Open Sans",sans-serif;Margin:0}
2 .Menu{Position:fixed;Width:18%}
3 .Menu:first-of-type{Top:0;Left:0}
4 Form{Margin:auto;Position:absolute;Width:64%;Top:5%;Left:0;Right:0}
5 Fieldset{Background-color:white;Margin:2em;Padding:2em}
6 .Menu:last-of-type{Height:13.5em;Margin:auto;Top:0;Bottom:0;Right:0}
```

Ukázka 5: Styly stránky a formuláře

Nejprve si nastavíme společné styly pro tělo stránky *body*. Do složených závorek jsem zadal barvu pozadí a barvu textu (parametry *background-color* a *color*). Jejich hodnotami může být buď anglický název barvy, nebo RGB hodnota barvy, kterou můžeme zadat buď v šestnáctkové soustavě (křížek a dvě číslice pro každou z barev) nebo v desítkové soustavě (heslo *rgb* a kulaté závorky, v nichž jsou hodnoty pro každou z barev odděleny čárkami). Dále jsem zadal typ písma (viz kapitola 3.4.4 Písmo) a na závěr jsem vynuloval přednastavené okraje stránky.

Navigaci jsem označil třídou *Menu*, proto styl pro navigaci můžeme nyní označit tečkou a názvem třídy. Ale pozor, u tříd záleží na velikosti písma a je tedy rozdíl mezi *.MENU* a *.menu*. Pro označení prvního a posledního menu jsem k *.Menu* přidal takzvanou pseudotřídu. Ta se vypíše dvojtečkou a zadaným heslem, v našem případě tedy *:first-of-type*, *:last-of-type* nebo třeba *:nth-of-type(n)*, přičemž místo *n* doplníme pořadové číslo prvku. Jelikož chceme, aby navigace zůstala uživateli viditelná vždy, i když uživatel odroluje na stránce níž, nastavíme parametr *position* na hodnotu *fixed*. Nyní ještě doplníme pozici těchto menu parametry vzdáleností od okrajů stránky *top*, *left*, *bottom* a *right*. Také můžeme doplnit parametry výšky a šířky *height* a *width*.

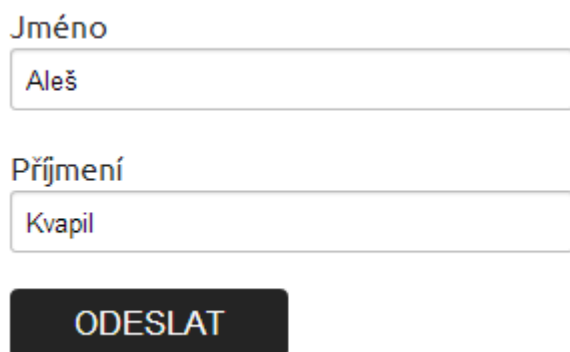
Všechny tyto vlastnosti se mohou nastavit v různých jednotkách. K dispozici máme pixely (px), centimetry a milimetry (cm, mm), relativní výšku základního velkého písma M (em) nebo malého písma x (ex), případně i procenta (%) a dalších zhruba dvanáct jednotek. Nejčastější jsou však ty výše vypsané a s nimi budeme pracovat.

Chceme-li nějaký prvek vycentrovat (ať už horizontálně, či vertikálně), použijeme drobný „hack“ ve formě automatických okrajů. Pro tento prvek nastavíme *margin*, prostor mezi prvkem a jeho obalem, na hodnotu *auto* a dále musíme nastavit vzdálenosti od okraje stránky ve směru, ve kterém chceme prvek vycentrovat, na nulovou hodnotu. Nesmíme zapomenout také nastavit rozměr prvku v daném směru (*height* nebo *width*). Ještě doplním, že chceme-li určitý parametr nastavit na nulovou hodnotu, nemusíme uvést žádnou jednotku. Nakonec je tu skrytá podmínka, že prvek musí mít také parametr *position* nastavený na jakoukoliv hodnotu kromě *relative*. Prvek vycentrovaný v horizontálním i vertikálním směru by tedy v kaskádových stylech měl nastaveno toto:

```
1 .Prvek{Margin:auto;Position:absolute;Height:50%;Width:50%;Top:0;Left:0;Bottom:0;Right:0}
```

Ukázka 6: Absolutně vycentrovaný prvek

Jak jsem již dříve uvedl, uživatelsky příjemný formulář by neměl působit příliš stísněně. Měli bychom tedy jednotlivým prvkům formuláře nastavit dostatečně veliké okraje. Mezi prvkem a jeho obsahem se o velikost okrajů stará parametr *padding*, mezi prvkem a jeho obalem zase výše zmíněný *margin*. Formulář s nastavenými okraji by pak mohl vypadat například takto:



The image shows a simple web form. At the top, there is a label 'Jméno' in a blue font. Below it is a text input field containing the text 'Aleš'. Below that is another label 'Příjmení' in a blue font, followed by a text input field containing the text 'Kvapil'. At the bottom of the form is a dark blue button with the text 'ODESLAT' in white capital letters.

Obrázek 2: Jednoduchý formulář s nastavenými styly

3.4.2 Formulářové prvky

Pro formulářová pole je časté zaoblování rohů, a stínování. Těchto efektů dosáhneme pomocí parametrů *border-radius* pro zaoblené rohy a *box-shadow* (případně *text-shadow*) pro stíny. Parametr pro stínování obsahuje posunutí ve vertikálním směru, posunutí v horizontálním směru a velikost rozmazání stínu. Může také obsahovat heslo *inset* na začátku, pokud potřebujeme nastavit vnitřní stín. Barva stínu se bere z barvy textu, můžeme ji však změnit přidáním libovolné barvy na závěr parametru. Všimněte si tedy například stínu pro text i samotný prvek u stylu pro *button:hover* a *button:active*. Pseudotřída *:hover* znamená, že se nad prvkem nachází myš (styly se použijí při najetí myši), *:active* zase znamená, že je prvek aktivován (je na něj kliknuto).

```
1 Button{Background-color:#242424;Border:0;Border-radius:0.25em;Color:white;Cursor:pointer;Font-size:1em;Margin:1em;Padding:1em;Text-transform:uppercase}
2 Button:hover,Button:active{Box-shadow:0 0 0.5em #242424;Text-shadow:0 0 0.5em}
3 Input,Textarea{Border:1px solid silver;Border-radius:0.25em;Box-shadow:inset 0 1px 1px rgba(0,0,0,0.1);Padding:0.5em}
4 .Kratky{Width:6em}
5 .Stredni{Width:12em}
6 .Dlouhy{Width:18em}
7 Textarea{Width:100%}
```

Ukázka 7: Různé styly formulářových prvků

Když už jsem zmínil vzhled tlačítek, bylo by dobré jejich vzhled ještě vylepšit. Můžeme například vynulovat nebo nastavit okrajovou linii tlačítka pomocí parametru *border*. Hodnota tohoto parametru sestává z tloušťky, stylu (plný, čerchovaný, apod.) a barvy. Jednotlivé hodnoty musíme oddělit mezerami. Všimněte si také parametrů *font-size* a *text-transform*. První určuje velikost písma, druhá zase velikost písmen. U *text-transform* máme k dispozici hodnoty *uppercase* pro velká písmena, *lowercase* pro malá písmena nebo *capitalize* pro první písmena velká. Nebo můžeme hodnotu vynulovat heslem *none*.

Můžete si ještě všimnout, že jsem tlačítku doplnil parametr *cursor* s hodnotou *pointer*, která zajistí, že kurzor myši nad tlačítkem bude „ukazující ruka“. Možná by mi někdo vytknul, že nad tlačítky se kurzor mění na ruku vždy. Pokud ovšem změníme vzhled tlačítka kaskádovými styly, musíme znovu nastavit také chování kurzoru. Zde jde o historicky podloženou chybu prohlížečů, raději se proto podřídíme.

Dále si povšimněte, že jsem nastavil délku textových polí. Ke každému vstupnímu textovému poli bude zapotřebí přiřadit jednu z nastavených tříd. Tímto jsem využil kaskádovosti jazyka CSS, který mi umožňuje nastavit jeden vzhled pro více prvků. Zároveň si můžete všimnout více prvků před složenou závorkou. Tyto prvky jsou od sebe odděleny čárkou a pro ně pak platí totožné styly.

3.4.3 Navigace

Jestliže chceme uživateli nabídnout plný uživatelský komfort, musíme mu umožnit se po našem formuláři jednoduše a rychle pohybovat. Proto bychom měli přidat nejen tlačítko na další stránku formuláře, ale také tlačítko zpět, menu pro přechod na libovolnou stránku a možnost kdykoliv odeslat formulář ke schválení nebo si data průběžně ukládat. Na tohle vše by nám měly posloužit tlačítka (`<button>`), případně odkazy (`<a>`).

Oblíbená a moderní jsou v dnešní době také tzv. „bullets“, čili kulatá tlačítka ve tvaru puntíku. Jejich vytvoření není nijak složité. V HTML místo názvu vepíšeme ` `, což je zástupný ASCII symbol pro nezlomitelnou mezeru (používanou v případech, kdy nechceme, aby se mezi slovy zalamoval řádek). Kdybychom tam vepsali klasickou mezeru nebo nechali text tlačítka prázdný, tlačítko by se vůbec nezobrazilo. Takto jsme si tedy vytvořili prázdné tlačítko a nyní nám chybí zaoblit jeho okraje dokulata.

```
1 .Menu:last-of-type Button{Background-color:white;Border-radius:8px;Height:8px;Padding:0;Width:8px}
2 .Menu:last-of-type Button:hover,.Menu:last-of-type Button:active{Background-color:#242424}
```

Ukázka 8: Kulatá tlačítka

Další funkcí Kaskádových stylů je možnost vybrat prvek, který je uvnitř prvku jiného. Používá se k tomu jednoduše mezera. V příkladu tedy vybírám všechna tlačítka z druhého menu a přiděluji jim styly. Chci-li dosáhnout kulatého tlačítka, musím mu nejprve nastavit stejnou šířku i výšku, a poté nastavit zaoblené rohy tak, aby bylo zaoblení minimálně stejně velké, jako šířka a výška. Nakonec už jenom vynuluji vnitřní okraje v tlačítku a přidám barvu pro samotné tlačítko i barvu pro aktivní tlačítko. Výsledek poté vypadá zhruba takto:



Obrázek 3: Kulatá tlačítka

3.4.4 Písmo

Slíbil jsem, že se na písmo a jeho vkládání podíváme blíže. Slib dodržím, ale upozorňuji, že je to jedna z náročnějších funkcí našeho formuláře. Možností pro vložení písma do stránky je samozřejmě více. Záleží na tom, v jakém formátu písmo máme uložené. Nejjednodušší k vložení jsou tzv. systémová písma (např.: Times New Roman

nebo Arial). Ta jsou součástí všech operačních systémů a můžeme je tedy rovnou použít v našich stylech pomocí parametru *font-family*:

```
1 Body{Font-family:Arial,sans-serif}
```

Ukázka 9: Vložení písma do souboru

Jednotlivá použitá písma se oddělují čárkami, víceslovné názvy písem se uzavírají do uvozovek. Roli zde hraje také velikost písmen, raději proto název písma přesně opište. Platí zde pravidlo, že se prohlížeč pokusí načíst první písmo, a pokud se mu to nepovede, přejde na další v řadě. Na závěr je tedy vhodné vložit jedno z klíčových slov *serif* (patkové), *sans-serif* (bezpatkové), *cursive* (stočené, psací), *fantasy* (dekorativní, expresivní) a *monospace* (se stejnou šířkou písmen, počítačové). Prohlížeč pak za dané klíčové slovo dosadí nejběžnější systémové písmo pro danou kategorii (např.: Times New Roman pro klíčové slovo *serif*).

Horší jsou však další případy, kdy chceme použít nesystémové písmo. Jestliže jsme takové písmo získali z online databází (například z Google Fonts), pak je u každého písma doplněný většinou i návod k použití, často i s předpřipraveným kódem. Ten pak stačí už jen vložit na příslušné místo a vše je hotovo.

Největší problém nám však bude dělat písmo, které máme k dispozici v souboru na našem počítači. Takové písmo je většinou ve formátu *.ttf* nebo *.otf*. Zde nejprve musíme písmo zkonvertovat do dalších formátů pomocí nějakého konvertoru. Písmo kvůli zpětné kompatibilitě totiž budeme potřebovat ve formátech *.eot*, *.svg*, *.woff* a *.ttf* nebo *.otf*. Takto připravené čtyři soubory poté musíme vepsat do našich stylů přesně podle zadané formule:

```
1 @font-face{
2   Font-family:'Moje pismo';
3   Src:url('mojepismo.eot?#iefix') format('embedded-opentype'),
4   url('mojepismo.woff') format('woff'),
5   url('mojepismo.ttf') format('truetype'),
6   url('mojepismo.svg') format('svg')
7 }
8 Body{Font-family:'Moje pismo',Arial,sans-serif}
```

Ukázka 10: Formule pro vložení písma do stránky

3.4.5 Prefixy

Na závěr uživatelské části se ještě musím zmínit o tzv. „prefixes“, čili předponách u jednotlivých stylů. Některé styly, které jsme použili výše, totiž nejsou zcela zpětně kompatibilní. Jedná se například o zaoblené rohy. U takových si poté musíme zjistit, jaké

předpony máme doplnit těmito stylům, aby fungovali i na některých starších prohlížečích. K tomu nám může posloužit například internetová stránka CanIUse.com, která obsahuje databázi všech parametrů Kaskádových stylů, které je nějakým způsobem třeba opravit kvůli zpětné kompatibilitě. Taková předpona sestává z pomlčky, jména prohlížeče, který tuto funkci podporuje ve starších verzích, pomlčky a poté již následuje klasický zápis stylu. Pro zaoblené rohy tedy plný prefixovaný kód bude vypadat následovně:

```
1 Button{-moz-border-radius:0.25em;-webkit-border-radius:0.25em;border-radius:0.25em}
```

Ukázka 11: Prefixované zaoblené rohy u tlačítka

4 Strojová část

Dosud jsme se zabývali strukturou a vzhledem formuláře. Ten by nyní měl vypadat uživatelsky i designově perfektně, stále nám však nefunguje ani jedno tlačítko. Nyní tedy přichází na scénu ta nejdůležitější část zajišťující jeho funkčnost.

4.1 PHP v HTML kódu

Abychom mohli do souboru umístit kód PHP, musíme jej od ostatního obsahu souboru (například od kódu HTML nebo od Kaskádových stylů) oddělit. K tomu nám slouží otevírací `<?php` a zavírací `?>` značky, mezi které poté umístíme náš kód. Vždy jej ale od těchto značek musíme oddělit alespoň jednou mezerou nebo novým řádkem. Samotný kód pak obsahuje jednotlivé příkazy, které oddělujeme středníkem.

```
1 <?php
2 // Kód PHP
3 ?>
```

Ukázka 12: vložení PHP kódu do HTML dokumentu

4.2 Úvod do sessions

Funkce *sessions* nám v PHP umožňují pracovat s daty mezi více stránkami. Jejich využití je tedy například u vícestránkového formuláře více než jasné. *Session* nám umožní průběžně ukládat data, dále je zpracovávat a doplňovat již zadaná data, či s nimi jinak nakládat. Každá *session* má navíc svoje vlastní unikátní *session id*, jehož pomocí budeme moct zadaná data třídit a řadit.

Pokud chceme s daty pracovat pomocí *sessions*, musíme nejprve *session* spustit. K tomu slouží příkaz `session_start()`, který nejenže spustí v daném souboru *session*, ale také získá *session* data z předchozí stránky. Je potřeba jej proto uvést v každém souboru, kde chceme s daty v *session* pracovat. Data odeslaná z formuláře se bohužel neposílají skrz *session*, nýbrž pomocí polí *POST* nebo *GET*. Proto je z těchto polí musíme nejprve do *session* přepsat. To uděláme pomocí příkazů `$_POST['jmeno']` nebo `$_GET['jmeno']`. Do proměnné poté data z takových polí přepíšeme takto: `$Jmeno=$_POST['jmeno']`. Obsah těchto proměnných se ale při odchodu z dané stránky smaže, proto je přepíšeme

do speciálních *session* proměnných. Náš kód tedy bude vypadat takto:
`$_SESSION['jmeno']=$_POST['jmeno'];`

Pokud máme velmi mnoho hodnot, které chceme z formuláře zpracovat, pak můžeme použít PHP pole. Názvy jednotlivých formulářových prvků můžeme přepsat do jednoho pole (*array*), a poté je zpracovat cyklem *foreach*. Kód v souboru *zpracovani.php* tedy zatím bude vypadat zhruba takto:

```
1 <?php
2 session_start();
3 $Pole=array("Jmeno","Prijmeni");
4
5 if(!empty($_POST)){
6     foreach($Pole as $Promenna){
7         if(!empty($_POST[$Promenna])){
8             $_SESSION[$Promenna]=$_POST[$Promenna];
9         }}
10 ?>
```

Ukázka 13: Zpracování velkého množství dat pomocí pole a *foreach* cyklu

V ukázce 13 máme pole v proměnné *\$Pole*, v němž jsme vypsali jména všech formulářových prvků, které chceme zpracovat. Ta ještě nesmíme zapomenout obalit uvozovkami a oddělit čárkami. Poté následuje *foreach* cyklus, který nám z pole vybere veškeré hodnoty a pro každou hodnotu přepíše data z *POST* do *session*. Nesmíme se ještě zapomenout ujistit, že pole *POST* není prázdné, tedy že uživatel do formuláře zadal nějaké hodnoty, obdobně také u jednotlivých prvků formuláře, abychom zbytečně nezapisovali prázdná data.

4.3 Limitace sessions

Problém však nastává, pokud má uživatel vypnutá *cookies* ve svém prohlížeči. *Sessions* jsou totiž jinými slovy *cookies* s nulovou platností, což znamená, že jsou data v *sessions* uchovávána pouze do doby, než je *session* ukončena nebo je vypnut prohlížeč. Jelikož však *sessions* díky svému krátkodobému charakteru nejsou pravé *cookies*, jedná se tak o chybu. Na tuto chybu naštěstí PHP pamatuje, proto pokud je v prohlížeči vypnuto používání *cookies*, lze předávání dat v *session* opravit jednoduchým způsobem. Ten spočívá v tom, že se v adrese url za znakem otazníku u všech odkazů, které vedou na stránku, kde ještě budeme s daty pracovat, uvede název *session* a *session id*. PHP nám celou opravu ještě zjednoduší, protože nám pro správně sestavený parametr předávaný v adrese url poskytuje speciální proměnnou *SID*. Tu už pak můžeme vložit do adresy url

pomocí funkce `echo()`, která jednoduše vypíše hodnotu proměnné nebo textu v uvozovkách. V našem případě tedy upravíme adresu url pro zpracování formuláře takto:

```
1 <Form Action="zpracovani.php?<?php echo SID; ?>" Id="Formular" Method="Post">
2 </Form>
```

Ukázka 14: Doplnění *SID* do adresy url

Jak jsem již dříve poznamenal u vlastnosti formuláře *method*, předávání parametrů pomocí metody *GET* přímo v adrese url má jistá bezpečnostní rizika. U předávání *session id* jsou tato rizika ještě vyšší. Pokud totiž útočník jakýmkoliv způsobem získá *session id* uživatele, má přístup ke všem jeho datům. Proto je lepší parametr *SID* psát do url pouze tehdy, když má uživatel opravdu vypnutá *cookies*. Toho docílíme například kombinací se skriptovacím jazykem, případně si můžeme přímo v PHP nastavit zkušební *session* nebo *cookie* proměnnou a při opětovném načtení stránky zjistit, jestli se nám tato proměnná uchovala, či nikoliv.

4.4 Zápis dat do cookies

Nyní již máme data přepsaná v *session* proměnných. Naším dalším cílem je tato data uložit a zapsat. Na výběr máme více cest. Data můžeme zapsat jak na server, tak do paměti prohlížeče. Pro zápis dat na server používáme nejčastěji *databáze*, v prohlížeči zase využijeme *cookies*.

Cookies zapisují data do paměti prohlížeče. Uživatel je tím pádem může omezit, či úplně zakázat, navíc je může kdykoliv smazat, což nám neumožňuje s nimi pracovat jako s hlavním úložným zařízením. Z pozice sběratele formulářů je také nemůžeme nijak číst. Proto do nich zapíšeme pouze dočasná data, která chceme, aby vydržela v paměti déle, než *session* data, a jejichž uplatnění bude sloužit jako záložní například v případě, že v průběhu vyplňování formuláře dojde k nějaké chybě, ať už na straně serveru nebo uživatele.

Zápis se provádí příkazem `setcookie()`. Ten má v závorce tři parametry oddělené čárkou. Prvním parametrem je jméno *cookie*, druhým její hodnota a třetím čas ve vteřinách od 1. ledna 1970, do kdy bude k dispozici. Pro třetí parametr udávající čas můžeme použít PHP proměnnou `time()`, která vypíše aktuální čas. K ní pak můžeme přičíst samotnou dobu, po kterou chceme data zachovat. Zápis hodnoty by pak mohl vypadat například takto:

```
3 setcookie("Jmeno", "Hodnota", time()+60*60*24);
```

Ukázka 15: Uložení jednoduché hodnoty do *cookie* na dobu jednoho dne

Větší množství dat zapíšeme podobně, jako jsme je zpracovali. Také využijeme *foreach* cyklu, *\$Promenna* stále zastává jednotlivá jména formulářových polí a do *cookies* se tedy zapíše jméno i hodnota.

```
3 foreach($Pole as $Promenna){  
4     if(!empty($_POST[$Promenna])){  
5         setcookie($Promenna,$_SESSION[$Promenna],time()+60*60*24);  
6     }  
}
```

Ukázka 16: Zápis většího množství dat hodnot do *cookies*

4.5 Zápis dat do databáze

Nejbezpečnější možností zápisu dat jsou bezpochyby serverové databáze. Důležité je mít na serveru databázi přednastavenou a mít připravenou tabulku s kolonkami, do kterých budeme data vkládat. Vše můžeme připravit buď pomocí jazyka SQL, nebo můžeme použít například nástroj *phpMyAdmin*.

S databází MySQL pak můžeme v PHP pracovat třemi různými způsoby. Prvním z nich je procedurální rozšíření PHP MySQL. Je to původní a nejstarší možnost, jak s databází pomocí PHP pracovat. Má to ale háček. Toto rozšíření od verze MySQL 4.1.3 neumožňuje používat některé nové funkce tohoto jazyka, proto není doporučeno jej používat (Moravec, 2010). Druhou možností je tzv. *PDO*, což je objektově orientovaná vrstva PHP pro práci s databázemi. Je vyvíjena tak, aby pomocí této vrstvy bylo možno pracovat s více druhy databází, nejenom s MySQL, avšak pro začátečníky není kvůli jejímu objektově orientovanému přístupu zcela vhodná. Poslední možností je rozšíření PHP MySQLi (MySQL Improved), což je vlastně původní rozšíření MySQL doplněné o nové funkce tohoto jazyka. Navíc umožňuje jak objektově orientovaný, tak procedurální přístup k databázi a v podstatě se mnoho neliší od původního rozšíření PHP MySQL.

Abychom mohli s naší databází pracovat, musíme se k ní nejprve připojit. Toho docílíme pomocí funkce *mysqli_connect()*, která má čtyři parametry. Tím prvním je adresa serveru, kde databáze běží, a následuje uživatelské jméno a heslo k připojení do databáze a jméno samotné databáze. Celou funkci je ještě vhodné vložit do proměnné, protože pomocí ní budeme dále k databázi přistupovat. Tato proměnná se nyní bude vkládat do prvního parametru všech příkazů souvisejících s naší databází.

```
3 $Database=mysqli_connect("localhost","uzivatelskejmeno","heslo","jmenodatabase");
```

Ukázka 17: Připojení k databázi

Dále musíme stanovit sadu znaků pro přístup k databázi. Toho dosáhneme pomocí funkce *mysqli_set_charset()*, jejímž prvním parametrem je právě funkce *mysqli_connect()*, a druhým parametrem pak je samotná sada znaků. Zápis sady znaků se zde ale řídí podle jazyka SQL, proto musíme sadu *windows-1250* změnit na *cp1250*, sadu *iso-8859-2* na *latin2* a *utf-8* na *utf8*.

```
3 mysqli_set_charset($Database,"utf8");
```

Ukázka 18: Nastavení sady znaků pro práci s databází

Teď již můžeme začít pracovat s databází. Nejprve sestavíme příkaz SQL obsahující data, která chceme do databáze zapsat, a poté tento příkaz zpracujeme pomocí funkce *mysqli_query()*. Příkaz SQL pro zapsání dat do tabulky databáze je *insert into*, za nímž následuje název tabulky. Další je příkaz *set*, za nímž se uvádí zapisovaná jednotlivá data. Ta se vkládají ve formátu *Sloupec='Hodnota'* a oddělují se čárkou. Celý samotný příkaz jazyka SQL by pak mohl vypadat nějak takto:

```
1 insert into Tabulka set Sloupec='Hodnota'
```

Ukázka 19: Příkaz jazyka SQL pro zápis do databáze

Ještě, než příkaz odešleme, musíme ale veškeré hodnoty zadané uživatelem přepsat do strojově čitelné hodnoty. Musíme tedy znaky, které jsou příkazy jazyka SQL nebo které nejsou zahrnuty v nastavené sadě znaků tzv. *escapovat* (čti „iskejnovat“), k čemuž slouží funkce *mysqli_real_escape_string()*. Jejím prvním parametrem je stále příkaz *mysqli_connect()* a druhým parametrem je SQL příkaz, který chceme přepsat. Proměnné, v nichž máme uložené hodnoty, které chceme do tabulky napsat, pak můžeme do příkazu vložit přímo dovnitř dvojitých uvozovek.

```
3 $Hodnota=mysqli_real_escape_string($Database,$Hodnota);  
4 $Prikaz="insert into Tabulka set $Sloupec='$Hodnota';"  
5 mysqli_query($Database,$Prikaz);
```

Ukázka 20: Zpracování SQL příkazu

Takto připravený SQL příkaz pak již můžeme zapsat do funkce *mysqli_query()*, jejíž parametry jsou totožné s funkcí *mysqli_real_escape_string()*. Na závěr ukončíme spojení s databází pomocí příkazu *mysqli_close()*. Ani zde nesmíme zapomenout na první parametr.



```
1 mysqli_close($Database);
```

Ukázka 21: Ukončení práce s databází

Pokud máme větší množství dat, která chceme zapsat, pak data do příkazu zpracujeme podobně, jako jsme to udělali u *cookies*. U cyklu ještě využijeme principu tzv. *string concatenation*, čili spojování více textů pomocí operátoru `.` (tečka). Příprava takového příkazu bude poté vypadat zhruba takto:

```
3 $Prikaz="insert into Tabulka set ";
4 foreach($Pole as $Promenna){
5     if(!empty($_POST[$Promenna])){
6         $_SESSION[$Promenna]=mysqli_real_escape_string($Database,$_SESSION[$Promenna]);
7         $Prikaz.="$_SESSION['{$_SESSION[$Promenna]}'],'";
8     }}

```

Ukázka 22: Příprava příkazu pro zápis většího množství dat do databáze

4.6 Zápis dat do souboru

Další možností, jak pracovat se zadanými daty uživatele, může být na serveru zápis do souboru. Můžeme jej využít jak k samotnému uložení dat, tak k vygenerování vyplněného formuláře pro uživatele ke stažení nebo vtištění. Budeme potřebovat funkce *fopen()*, *fwrite()* a *fclose()*, které k danému účelu slouží. Pomocí *fopen()* vytvoříme soubor nebo jej otevřeme, pokud již byl vytvořen, pomocí *fwrite()* do něj zapíšeme data a pomocí *fclose()* ukončíme práci se souborem.

Funkce *fopen()* má v závorce dva hlavní parametry. Tím prvním je jméno souboru, případně cesta k němu, tím druhým parametr určující, k jakým operacím budeme chtít soubor použít. Na výběr máme z více možností. Ke čtení souboru používáme nejčastěji parametrů *r* pro otevření stávajícího souboru a nastavení kurzoru na jeho začátek, *w* pro vytvoření nového souboru nebo vymazání obsahu stávajícího a *a* pro vytvoření nového souboru nebo načtení stávajícího, přičemž kurzor se nastaví na konec souboru. Ke každému parametru se ještě přidává `+`, pokud chceme soubor otevřít i k zápisu. K uložení dat je tedy vhodné použít *a+*, k vygenerování unikátního formuláře pro každého uživatele pak použijeme *w+* v kombinaci s unikátním jménem souboru (zde můžeme například využít *session id*).

Funkce *fwrite()* má také dva parametry. Prvním z nich je funkce *fopen()*, kterou jsme si již dříve uložili do proměnné. Druhým parametrem bude text, který chceme zapsat. I zde

můžeme spojit více textů a proměnných do jednoho pomocí tečky. Funkce *fclose()* pak má jenom jeden parametr totožný s prvním parametrem u *fwrite()*.

```
3 $Soubor=fopen(session_id()).".html","a+");
4 fwrite($Soubor,"Text, který chci zapsat".PHP_EOL);
5 fclose($Funkce);
```

Ukázka 23: Jednoduchý zápis do souboru

Pokud chceme do souboru data pouze uložit, pak nám postačí jenom textový vstup obsahující zapisované proměnné a obalený v uvozovkách. Pokud chceme v zapisovaném textu vytvořit nový řádek, použijeme PHP proměnnou *PHP_EOL*. Pokud však chceme uživateli vygenerovat vyplněný formulář k vytisknutí, pak bude jednodušší nahrát do *fwrite()* předpřipravenou šablonu uloženou v externím souboru. Abychom zpracovali proměnné, vkládané do této šablony, a zároveň nahráli obsah souboru do *fwrite()*, musíme použít funkci *include()*, jejímž parametrem je jméno souboru šablony, případně cesta k němu. Funkci *include()* jako takovou však nemůžeme nahrát do proměnné, proto použijeme tzv. *output buffering*. Pomocí této funkce jazyka PHP vytvoříme dočasně druhý paralelní běh PHP kódu uvnitř souboru, jakousi vyrovnávací paměť, která je nezávislá na zbytku kódu, ale dovede s okolím pracovat. Vytvoříme si tak jakousi „oddělenou místnost“. V ní nahrajeme obsah šablony pomocí *include()*, čímž jej automaticky zpracujeme (proměnné se nahradí jejich hodnotami) a poté celý obsah naší oddělené místnosti uložíme do proměnné.

```
3 ob_start();
4 include "sablon.php";
5 $Sablon=ob_get_contents();
6 ob_end_clean();
```

Ukázka 24: Nahrání obsahu šablony do proměnné pomocí *output buffering*

Oddělenou místnost uvedeme pomocí konstruktoru *ob_start()*. Následující kód až do uvedení konstruktoru *ob_end_**() nebo do konce samotného souboru už bude uvnitř oddělené místnosti. Nyní tedy můžeme doplnit funkci *include()*. Pomocí konstruktoru *ob_get_contents()* nahrajeme obsah oddělené místnosti, čili obsah vložené a zpracované šablony do proměnné a pomocí konstruktoru *ob_end_clean()* ukončíme oddělenou místnost a vymažeme dočasná data v ní. Proměnnou pak už můžeme vložit do příslušného parametru funkce *fwrite()*, případně s ní dále pracovat.

4.7 Odeslání potvrzujícího emailu

Obdobně připravíme a odešleme potvrzovací email. Znovu můžeme použít *output buffering* společně s předpřipravenou šablonou, pokud chceme vytvořit složitější email pomocí HTML a Kaskádových stylů. Při přípravě šablony emailu ale máme omezenější možnosti. Email by měl být napsán v jednoduché formě HTML s použitím co nejjednodušších stylů a nejlépe bez použití skriptovacích jazyků. Veškeré vložené soubory, jako například obrázky nebo videa, by měly být vloženy do emailu přímo, například pomocí konstrukce *data*. Pro naše účely však postačí i jednoduchý email bez obrázků.

Podstatnější limitací pro nás však bude, že emailová zpráva nesmí obsahovat více než 70 znaků na řádek, aby byla správně odeslána a přečtena. Užitečnou funkcí, díky které nebudeme muset při přípravě šablony emailu myslet na délku řádku, je funkce *wordwrap()*, která vždy po zadaném počtu znaků vloží zadaný oddělovač, například nový řádek. Do této funkce bychom měli zadat tři parametry. Prvním je zpráva, kterou chceme zpracovat, druhým je počet znaků a třetím je znak, který se vždy po zadaném počtu znaků vkládá do zprávy. Emailová zpráva by měla vždy obsahovat kompletní tzv. *CRLF* deklaraci, nemůžeme proto použít proměnnou *PHP_EOL*, která se mění v závislosti na operačním systému. V našem případě by tedy mohla funkce *wordwrap()* vypadat nějak takto:

```
3 $Zprava=wordwrap($Zprava,70,"\r\n");
```

Ukázka 25: Zpracování emailové zprávy

Samotné odeslání emailu pak provedeme pomocí funkce *mail()*. Jejími čtyřmi nejčastěji používanými parametry jsou postupně email příjemce, předmět, samotná zpráva a hlavička. Do hlavičky bychom měli zapsat *MIME* deklaraci, použitou sadu znaků (viz kapitola 3.1 Základní struktura formuláře) a odesílatele zprávy a oddělit jednotlivé položky hlavičky odřádkováním *CRLF*. Celá funkce *mail()* by pak mohla vypadat zhruba takto:

```
3 $Hlavicka="MIME-Version: 1.0"."r\n"
4 ."Content-type: text/html; charset=utf-8"."r\n"
5 ."From: email@odesilatele.cz";
6 mail("email@prijemce.cz","Předmět","Zpráva",$Hlavicka);
```

Ukázka 26: Odeslání potvrzovacího emailu

4.8 Získání zapsaných dat

Pokud uživatel odešle, uloží nebo jinak opustí formulář (například neúmyslně chybou serveru nebo klienta), měl by mu náš formulář umožnit se k těmto datům vrátit,

získat je zpět, obnovit je. Ukázali jsme si ukládání do *sessions*, *cookies*, databáze nebo souboru. Získání dat zpět je až na výjimky obdobné způsobu, jakým jsme data zapisovali.

Nejjednodušší je získat data ze *session*. Jestliže ji uživatel má stále spuštěnou a nevypnul v mezech prohlížeč, pak ověříme, zda jsou data v *session* ještě dostupná a nahrajeme je zpět. K nahrání dat zpět do formulářových polí stačí do hodnoty *value* u prvku `<input>` vložit samotnou *session* proměnnou pomocí funkce *echo()*. Abychom zamezili tomu, že se nám do prvku formuláře vepíše chyba, zjistíme pomocí podmínky *if*, zda daná proměnná existuje nebo zda není prázdná. První možnost obstarává funkce *isset()*, druhou funkce *empty()*. Jejich společným jediným parametrem je proměnná, kterou chceme vyšetřit a použití té, či oné funkce je čistě na našem rozhodnutí. Poté vypíšeme buď hodnotu proměnné, nebo prázdnou hodnotu (prázdnými uvozovkami `""`). Pro takto jednoduchou podmínku stačí použít podmínkové operátory `?:` jejichž parametry vyjadřuje nejlépe toto schéma: *podmínka?platí:neplatí*. Zápis takovéto zkrácené podmínky a vypsání různých hodnot pro různé případy by v našem případě vypadalo zhruba takto:

```
1 <?php echo !empty($_SESSION["Jmeno"])?$_SESSION["Jmeno"]:""; ?>
```

Ukázka 27: Podmínka pro vypsání hodnoty uložené v *session*, pokud existuje

Podobnou podmínkou se dají vložit i data z *cookies*. Jediným rozdílem vlastně je, že místo `$_SESSION["Jmeno"]` použijeme `$_COOKIE["Jmeno"]`. Pro přepsání dat z *cookies* zpět do nové *session* můžeme použít stejného *foreach* cyklu a stejného pole, jakým jsme dříve hodnoty zapisovali do *cookies*, ovšem nyní místo funkce *setcookie()* stačí jenom příkaz `$_COOKIE[$Promenna]`.

Složitější už ale bude získání dat z databáze nebo souboru. Pro potřeby prohlédnutí si zapsaných uživatelů sběratelem formulářů stačí, pokud si data v databázi prohlédneme pomocí aplikace *phpMyAdmin*, nebo pokud si soubor s daty otevřeme. Pokud ale potřebujeme data vložit uživateli zpět do formuláře, pokud například uživatel potřebuje data po určité době aktualizovat, musíme data načíst pomocí PHP.

K získání dat z databáze bude znovu potřeba se nejprve k databázi připojit, nastavit sadu znaků, znovu odeslat připravený příkaz a ukončit spojení. Jediný rozdíl bude spočívat v tom, že samotný příkaz bude vypadat trochu jinak. Příkaz pro vybrání určité hodnoty z tabulky je *select*. Za ním následuje výčet čárkami oddělených sloupců, ze kterých chceme získat data. Chceme-li získat data ze všech sloupců, pak výčet sloupců stačí

nahradiť hvězdičkou. Tu však pro naše účely nebudeme potřebovat. Dalším příkazem *from* vybereme příslušnou tabulku, ze které chceme získat data. Poslední příkaz *where Sloupec='Hodnota'* nám pak vybere pouze ty řádky, na kterých se ve *Sloupci* vyskytuje zadaná *Hodnota*. Celý samotný SQL příkaz pak bude vypadat takto:

```
1 select Jmeno from Tabulka where Prijmeni='Burda'
```

Ukázka 28: SQL příkaz pro získání určité hodnoty z databáze

Data ze souboru získáme a nahrajeme do proměnné podobným způsobem, jako u šablony. Můžeme použít původní *output buffering*, ale protože nyní již nepotřebujeme zpracovat žádné proměnné, tak jako tomu bylo u šablony, postačí nám funkce *file_get_contents()*, jejíž parametr je stejný jako u funkce *include()*. Pomocí této funkce vložíme obsah souboru přímo do proměnné a budeme moci se souborem dále pracovat. K dalšímu zpracování poté použijeme mnoho užitečných *string* funkcí. Poslouží nám například *explode()*, *strtok()*, *parse_str()*, *similar_text()*, *str_split()*, *strip_tags()*, *strpos()*, *strstr()*, *nl2br()*, *wordwrap()*, *preg_match()*, *substr()* a mnoho dalších.

4.9 Vícenásobné vkládání kódu

Jazyk PHP ale nemusíme využít jenom k práci s daty ve formuláři. Můžeme si jím usnadnit například přípravu formuláře samotného. K tomuto účelu využijeme funkcí, které jsme si již představili dříve. Využijeme polí, *if* podmínek i *foreach* cyklů.

Pro příklad můžeme chtít vytvořit větší množství textových formulářových polí. Do každého pole budeme chtít ještě automaticky doplnit hodnotu, pokud již je uložena v *session*. A ke každému poli budeme chtít doplnit navíc popisek. V samotném HTML by jedno takové pole vypadalo takto:

```
1 <Label For="Jmeno">Jméno</Label>
2 <Br>
3 <Input Id="Jmeno" Name="Jmeno" Tabindex="1" Type="Text" Value="<?php
4 echo !empty($_SESSION['Jmeno'])?$_SESSION['Jmeno']:' ';
5 ?>">
6 <Br>
```

Ukázka 29: Kompletní formulářové textové pole v HTML

Takových polí pak ve formuláři může být nespočet. Vypisovat všechna taková pole by bylo neefektivní a časově náročné. PHP nám však v tomto procesu umí dobře pomoci. Začneme tedy tím, že si vytvoříme pole, do kterého vypíšeme všechny hodnoty, které se

nám budou měnit. Pokud je takových hodnot více než jedna, pak můžeme do polí místo hodnot samotných vepsat další vnořená pole. V našem případě se mění tři proměnné: *id*, *tabindex* a *name*, případně další proměnné mající stejnou hodnotu jako *name*. Do každého vnořeného pole tedy vložíme tři různé hodnoty. Celé pole pak vypadá zhruba takto:

```
1 <?php
2 $Pole=array(
3   array(1,"Jméno","Jméno"),
4   array(2,"Příjmení","Příjmení"),
5   array(3,"Titul","Titul"),
6   //...
7 );
8 ?>
```

Ukázka 30: Pole s vnořenými poli a proměnnými

Další postup již důvěrně známe. Pomocí *foreach* cyklu a doplňkových *if* podmínek vygenerujeme celý formulář a dosadíme do něj jednotlivé hodnoty. Určitou hodnotu z pole pak můžeme získat pomocí hranatých závorek, do nichž se zapíše číslo nebo název příslušné hodnoty. Například první hodnotu v poli získáme pomocí příkazu *\$Pole[0]*. Nesmíme zde zapomenout na fakt, že číslování v programovacích jazycích začíná vždy od nuly, nikoliv od jedničky. Pokud chceme z ukázky 30 vybrat z vnořeného pole hodnotu *Příjmení*, pak náš příkaz napíšeme jako *\$Pole[1][2]*. S pomocí *foreach* cyklu pak můžeme celý proces zautomatizovat a vypsát tolik proměnných, kolik vnořených polí s hodnotami máme. Získání hodnot *Jméno*, *Příjmení* a *Titul* uvnitř cyklu tedy provede příkaz *\$Promenna[2]*. Celý cyklus by pro tento případ mohl vypadat třeba takto:

```
1 <?php
2 foreach($Pole as $Promenna){
3   echo "<Label For='".$Promenna[1]."'>".$Promenna[2]."</Label><Br>".PHP_EOL
4   . "<Input Id='".$Promenna[1]."' Name='".$Promenna[1]."' Tabindex='".$Promenna[0]
5   . "' Type='Text' Value='".(empty($_SESSION[$Promenna[1]])?$_SESSION[$Promenna[1]]:"")
6   . "'><Br>".PHP_EOL;
7 }
8 ?>
```

Ukázka 31: *foreach* cyklus pro vypsání většího množství formulářových prvků

4.10 Detekce a oprava chyb

Poslední důležitou částí, o které bych se měl zmínit, je detekce chyb a jejich řešení. Chyba může nastat v jakékoliv části našeho kódu. Měli bychom na tento fakt pamatovat a pokaždé, kdy by mohla chyba vzniknout, bychom se měli na její případný výskyt připravit. Pokud tedy například máme v kódu podmínku *if*, která zjišťuje, jestli není pole *POST* prázdné, tak bychom měli vždy uvést také podmínku *else*, pomocí níž bychom měli

uživateli zobrazit chybovou zprávu, pokud zadaná podmínka neplatí, případně ještě přerušit další zpracování kódu příkazem *break*, pokud se jedná o závažnou chybu. Kompletní podmínka by pak mohla vypadat například takto:

```
3 if(!empty($_POST)){
4 // Podmínka platí, pokračujeme
5 }else{
6 // Podmínka neplatí, vypíšeme chybu
7 echo "Chyba! Formulář nebyl vyplněn";
8 break;
9 }
```

Ukázka 32: Jednoduchá chybová hláška a přerušení zpracování kódu

Případně podobná podmínka zkráceným zápisem:

```
3 echo !empty($_POST["Jmeno"])?$_POST["Jmeno"]:"Chyba! Jméno nebylo zadáno!";
```

Ukázka 33: Zkrácený zápis podmínky s chybovou hláškou

Chyba však nemusí nastat jenom v našem PHP kódu. Může se například vyskytnout při zpracování HTTP požadavku. Při výskytu takové chyby pak prohlížeč automaticky zobrazí uživateli vlastní chybovou stránku. Jenže stránka, kterou prohlížeč zobrazí, může být pro uživatele matoucí a nemusí správně detekovat a reflektovat nastalou chybu. Z tohoto důvodu bychom měli vytvořit vlastní chybovou stránku. Ta pak může obsahovat například omluvu, možnou příčinu chyby nebo třeba volby pro uživatele, jakým způsobem pokračovat.

Chyby přenosu HTTP mohou být dvojího typu. Jedny jsou způsobeny chybou serveru, druhé chybou HTTP požadavku. Chyba serveru může být například chyba 503, kdy server neodpovídá na požadavek klienta. Častou chybou HTTP požadavku pak je chyba 404, pokud uživatel žádal adresu url, která neexistuje, ať už kvůli překlepu, nebo kvůli chybnému odkazu.

K detekci těchto chyb a přesměrování uživatele na naší chybovou stránku si vytvoříme soubor s názvem *.htaccess*. Ten nahrajeme do kořenového adresáře našeho webu. Do tohoto souboru pak napíšeme kód chyby a relativní cestu k vlastní chybové stránce. Obsah souboru by pak mohl vypadat například takto:

```
1 ErrorDocument 404 /404.php
2 ErrorDocument 503 /503.php
```

Ukázka 34: Soubor *.htaccess*

Pokud chceme zjistit, kde vzniklá chyba nastala, poslouží nám samozřejmě kód chyby. Ten však neposkytuje žádné detailní informace. Další informaci o chybě můžeme například zjistit z adresy url předešlé navštívené stránky. Tuto adresu získáme z PHP proměnné `$_SERVER["HTTP_REFERER"]`. Pokud je adresa prázdná, uživatel se při zadávání adresy url pravděpodobně překlepl. Pokud ale adresa prázdná není, najdeme v ní pravděpodobně chybný odkaz. Dalšími PHP proměnnými, které nám ještě lépe pomohou v rozpoznání vzniklé chyby, pak mohou být `$_SERVER["REQUEST_URI"]`, `$_SERVER["REQUEST_METHOD"]` nebo například `$_SERVER["REQUEST_TIME"]`, které zjistí požadovanou url, metodu předání dat nebo dobu, jak dlouho se požadavek zpracovával. Můžeme pak zjistit například, jakou adresu url uživatel požadoval, jestli se data ve formuláři odeslala pomocí správné metody nebo jestli není server přetížen. Tyto hodnoty tedy můžeme zpracovat, případně je uložit do souboru, či si je odeslat emailem, abychom je poté mohli posoudit a případnou chybu na naší straně napravit.



Obrázek 4: Jedna z nejpopulárnějších chybových stránek internetu

5 Závěr

Cílem mojí maturitní práce bylo naučit člověka s minimálními znalostmi o programování webových stránek, jak vytvořit přehledný a provozuschopný formulář. Podnětem k tomuto tématu mi byl fakt, že i přes svoji jednoduchost a uživatelskou přístupnost nejsou stále ještě webové formuláře v Čechách příliš rozšířené. Již v úvodu jsem naznačil, že tento jev může být způsoben nejen neznalostí programování a nedostatkem finančních prostředků k zaplacení programátora, který by se o vytvoření formuláře postaral, ale také zaleknutím se možné složitosti tvorby takového formuláře. Snažil jsem se proto v co nejjednodušší a laicky přístupné formě tento důvod k zaleknutí vyvrátit a poskytnout čtenáři komplexní „kuchařku“ k vytvoření nejčastějších funkcí webových formulářů.

Nebyl bych však úplně spokojen, kdybych si neověřil úspěšnost svojí práce. Mohl bych sice použít pro otestování sebe sama, protože jsem se v průběhu práce poznal mnoho nových funkcí programovacího jazyka PHP, naučil jsem se základy jazyka SQL, s nímž jsem dosud neměl možnost pracovat, a vytvořil jsem přehledný a uživatelsky přístupný webový formulář k přihlášení na vysokou školu, z něhož jsem použil ukázky kódu a jehož kód i vzhled samotný je k nahlédnutí v přílohách 1 a 2. Jenže já jsem již předem mnoho funkcí znal, samotné vytvoření HTML i CSS mi také nedělalo příliš velký problém a s tvorbou webových formulářů mám dostatek předchozích zkušeností.

Proto jsem požádal svého bratrance, který letos skládá maturitní zkoušku na střední škole zaměřené na polygrafii, jestli by se podle mého návodu nepokusil vytvořit libovolný jednoduchý formulář. I přes jeho počáteční obavy nakonec moji žádost přijal a s jeho středoškolskými základy programování webových stránek se směle pustil do práce. Díky jeho ochotě jsem zjistil, že moje práce úspěšně splnila svůj cíl, ale také jsem mohl svoji práci opravit i doplnit v místech, která pro něj nebyla dostatečně srozumitelná.

Doufám tedy, že moje práce bude nejen návodem jako takovým, ale také inspirací programátorům k psaní vlastních návodů a „kuchařek“, abychom smazali bázeň z očí začátečníků a předali svoje zkušenosti dále. Protože programování, pokud jej do vás tedy povinně doslova „nehustí“ na vysoké škole, stále stojí stranou klasických IT technologií právě kvůli těmto předsudkům a také kvůli nedostatku kvalitních návodů zaměřených na laiky, jichž je v Čechách stále bohužel většina.

Seznam zdrojů

Berjon, Robin, a další, [editor]. 2014. *HTML5*. 2014.

Fontspring. 2011. Fontspring. *The New Bulletproof @Font-Face Syntax*. [Online] 3. 2 2011. [Citace: 25. 3 2014.] <http://www.fontspring.com/blog/the-new-bulletproof-font-face-syntax>.

Frickey, Dean. 2008. A List Apart. *A More Useful 404*. [Online] 18. 11 2008. [Citace: 25. 3 2014.] <http://alistapart.com/article/amoreuseful404>.

Jahoda, Bohumil. 2013. Je čas. *Automatické zapamatování formulářů*. [Online] 24. 9 2013. [Citace: 25. 3 2014.] <http://jecas.cz/zalohovani-formularu>.

Michálek, Martin. 2014. Twitter. *Fontokuk*. [Online] 14. 2 2014. [Citace: 25. 3 2014.] <https://twitter.com/machal/status/434259484052295680>.

Molhanec, Michal. 2010. SourceForge. *MySQL (4.1 a vyšší) a čeština minifaq*. [Online] 30. 4 2010. [Citace: 25. 3 2014.] http://mol1111.users.sourceforge.net/mysql_cestina_minifaq.html.

Moravec, Zdeněk. 2010. Programujte.com. *PHP a MySQL – MySQLi – 1. díl*. [Online] 14. 1 2010. [Citace: 25. 3 2014.] <http://programujte.com/clanek/2009103100-php-a-mysql-mysqli-1-dil/>.

Oracle. 2014. *MySQL 5.6 Reference Manual*. 2014.

Ott, Michael. 2012. That Web Guy. *15 steps towards better form usability*. [Online] 2. 1 2012. [Citace: 25. 3 2014.] <http://www.thatwebguyblog.com/post/15-steps-towards-better-form-usability/>.

Příspěvatelé Wikipedie. 2014. Wikipedie. *Apache HTTP Server*. [Online] 10. 2 2014. [Citace: 25. 3 2014.] http://cs.wikipedia.org/w/index.php?title=Apache_HTTP_Server&oldid=11200224.

—. 2014. Wikipedie. *Polyfill*. [Online] 11. 3 2014. [Citace: 25. 3 2014.] <http://en.wikipedia.org/w/index.php?title=Polyfill&oldid=599178659>.

—, 2014. Wikipedie. *PHP*. [Online] 8. 3 2014. [Citace: 25. 3 2014.] <http://cs.wikipedia.org/w/index.php?title=PHP&oldid=11280702>.

—, 2014. Wikipedie. *Document Object Model*. [Online] 8. 2 2014. [Citace: 25. 3 2014.] http://cs.wikipedia.org/w/index.php?title=Document_Object_Model&oldid=11187461.

—, 2013. Wikipedie. *Zpětná kompatibilita*. [Online] 10. 3 2013. [Citace: 25. 3 2014.] http://cs.wikipedia.org/w/index.php?title=Zp%C4%9Btn%C3%A1_kompatibilita&oldid=9881042.

Spear, Cameron. 2011. Cameron Spear. *Get a Single Value from MySQL Database in One Line of PHP*. [Online] 2. 10 2011. [Citace: 25. 3 2014.] <http://cameronspear.com/blog/get-a-single-value-from-mysql-database-in-one-line-of-php/>.

Vetešník, Oldřich. 2013. Zdroják. *Formuláře v HTML5 a nové atributy*. [Online] 25. 3 2013. [Citace: 25. 3 2014.] <http://www.zdrojak.cz/clanky/formulare-v-html5-a-nove-atributy/>.

Vrana, Jakub, a další. 2014. *PHP Manual*. [editor] Philip Olson. 2014.

Walsh, David. 2007. David Walsh. *PHP Cookies: How to Set Cookies & Get Cookies*. [Online] 2. 10 2007. [Citace: 25. 3 2014.] <http://davidwalsh.name/php-cookies>.

Wu, Tiffany. 2013. Tiffbits. *CSS3 Form Styling Cheat Sheet*. [Online] 19. 12 2013. [Citace: 25. 3 2014.] <http://code.tiffbits.com/css3-form-styling-cheat-sheet/>.

Zajíc, Petr. 2005. Linux Software. *MySQL (1) - pestrý svět databází*. [Online] 1. 3 2005. [Citace: 25. 3 2014.] http://www.linuxsoft.cz/article.php?id_article=731.

Seznam obrázků

Obrázek 1: Jednoduchý formulář bez nastavených stylů 16

Obrázek 2: Jednoduchý formulář s nastavenými styly 18

Obrázek 3: Kulatá tlačítka.....	20
Obrázek 4: Jedna z nejpoužívanějších chybových stránek internetu	35

Seznam ukázek kódu

Ukázka 1: Základní struktura HTML stránky	12
Ukázka 2: Hlavička stránky	13
Ukázka 3: Tělo stránky	14
Ukázka 4: Jednotlivé prvky formuláře	15
Ukázka 5: Styly stránky a formuláře.....	17
Ukázka 6: Absolutně vycentrováný prvek	18
Ukázka 7: Různé styly formulářových prvků	19
Ukázka 8: Kulatá tlačítka	20
Ukázka 9: Vložení písma do souboru	21
Ukázka 10: Formule pro vložení písma do stránky	21
Ukázka 11: Prefixované zaoblené rohy u tlačítka.....	22
Ukázka 12: vložení PHP kódu do HTML dokumentu	23
Ukázka 13: Zpracování velkého množství dat pomocí pole a <i>foreach</i> cyklu	24
Ukázka 14: Doplnění <i>SID</i> do adresy url	25
Ukázka 15: Uložení jednoduché hodnoty do <i>cookie</i> na dobu jednoho dne	26
Ukázka 16: Zápis většího množství dat hodnot do <i>cookies</i>	26
Ukázka 17: Připojení k databázi	27

Ukázka 18: Nastavení sady znaků pro práci s databází	27
Ukázka 19: Příkaz jazyka SQL pro zápis do databáze.....	27
Ukázka 20: Zpracování SQL příkazu.....	27
Ukázka 21: Ukončení práce s databází	28
Ukázka 22: Příprava příkazu pro zápis většího množství dat do databáze	28
Ukázka 23: Jednoduchý zápis do souboru	29
Ukázka 24: Nahrání obsahu šablony do proměnné pomocí <i>output buffering</i>	29
Ukázka 25: Zpracování emailové zprávy.....	30
Ukázka 26: Odeslání potvrzovacího emailu	30
Ukázka 27: Podmínka pro vypsání hodnoty uložené v <i>session</i> , pokud existuje.....	31
Ukázka 28: SQL příkaz pro získání určité hodnoty z databáze	32
Ukázka 29: Kompletní formulářové textové pole v HTML.....	32
Ukázka 30: Pole s vnořenými poli a proměnnými	33
Ukázka 31: <i>foreach</i> cyklus pro vypsání většího množství formulářových prvků.....	33
Ukázka 32: Jednoduchá chybová hláška a přerušení zpracování kódu.....	34
Ukázka 33: Zkrácený zápis podmínky s chybovou hláškou	34
Ukázka 34: Soubor <i>.htaccess</i>	34

Seznam příloh

Příloha 1: Snímky vzhledu výsledného formuláře

Příloha 1: Snímky vzhledu výsledného formuláře

Součástí této maturitní práce bylo i vytvoření jednoduché a přehledné přihlášky na vysokou školu. Vytvářel jsem tuto přihlášku průběžně se psaním této práce a ukázky kódu jsem vkládal přímo do textu. Výsledek je nyní umístěn na serveru a je možné si veškeré funkce kdykoliv vyzkoušet. Níže jsou vybrané snímky tohoto formuláře, ukazující vzhled, funkcionalitu a další možnosti, které jsem v tomto formuláři využil.

Seznam snímků:

Snímek 1: Vstupní stránka přihlášky

Snímek 2: Jednoduchá stránka formuláře

Snímek 3: Formulářové pole pro vložení času a data (funkční pouze na novějších prohlížečích)

Snímek 4: Tabulka s možností vytvoření dalších řádků

Snímek 5: Kontrola zadaných dat k odeslání přihlášky

Snímek 6: Stránka informující o úspěšném odeslání přihlášky

Snímek 7: Stránka informující o uložení rozpracované přihlášky do cookies prohlížeče

Snímek 8: Vstupní obrazovka pro zobrazení údajů o odeslané přihlášce

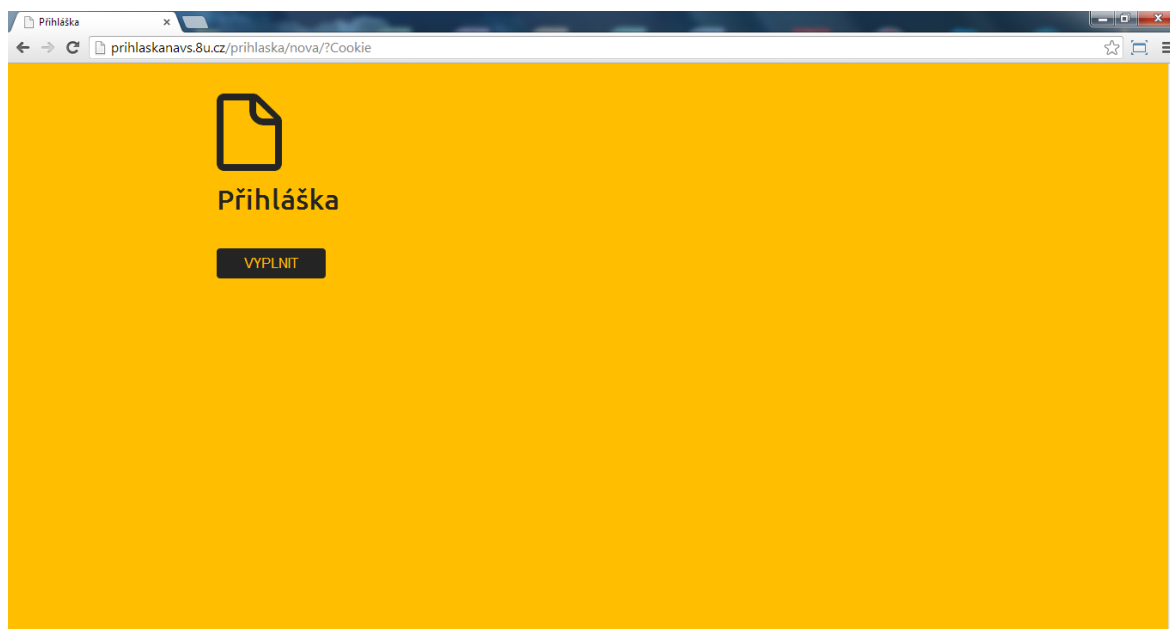
Snímek 9: Tabulka zobrazující údaje o odeslané přihlášce

Snímek 10: Vstupní stránka pro načtení uložené rozpracované přihlášky

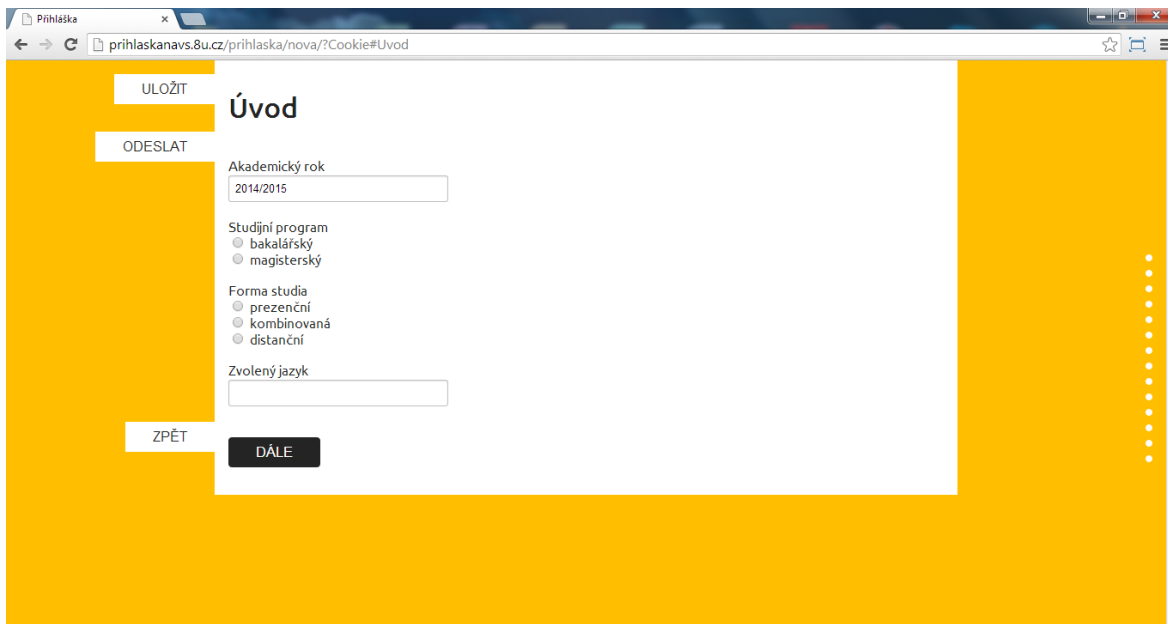
Snímek 11: Chybová stránka

Snímek 12: Práce v textovém editoru Brackets s ukázkou části souboru *zpracovani.php*

Snímek 1: Vstupní stránka přihlášky



Snímek 2: Jednoduchá stránka formuláře





Snímek 3: Formulářové pole pro vložení času a data (funkční pouze na novějších prohlížečích)

Přihláška x

prihlaskanavs.8u.cz/prihlaska/nova/?Cookie#Narozeni

ULOŽIT

ODESLAT

Narození

Datum narození

dd.mm.yyyy

březen 2014

po	út	st	čt	pá	so	ne
24	25	26	27	28	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

Rodné číslo

Číslo pasu (u cizinců)

ZPĚT

DÁLE

Snímek 4: Tabulka s možností vytvoření dalších řádků

Přihláška x

prihlaskanavs.8u.cz/prihlaska/nova/?Cookie#Prospech

ULOŽIT

ODESLAT

Prospěch

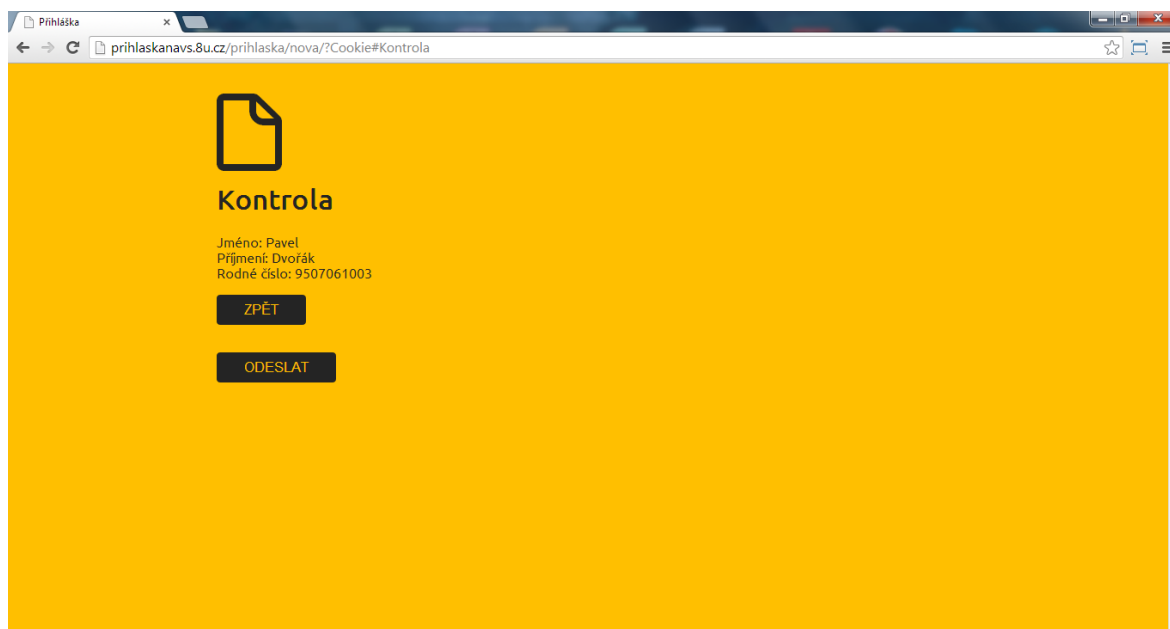
Předmět	Ročník					Maturita
	I	II	III	IV	V	

+

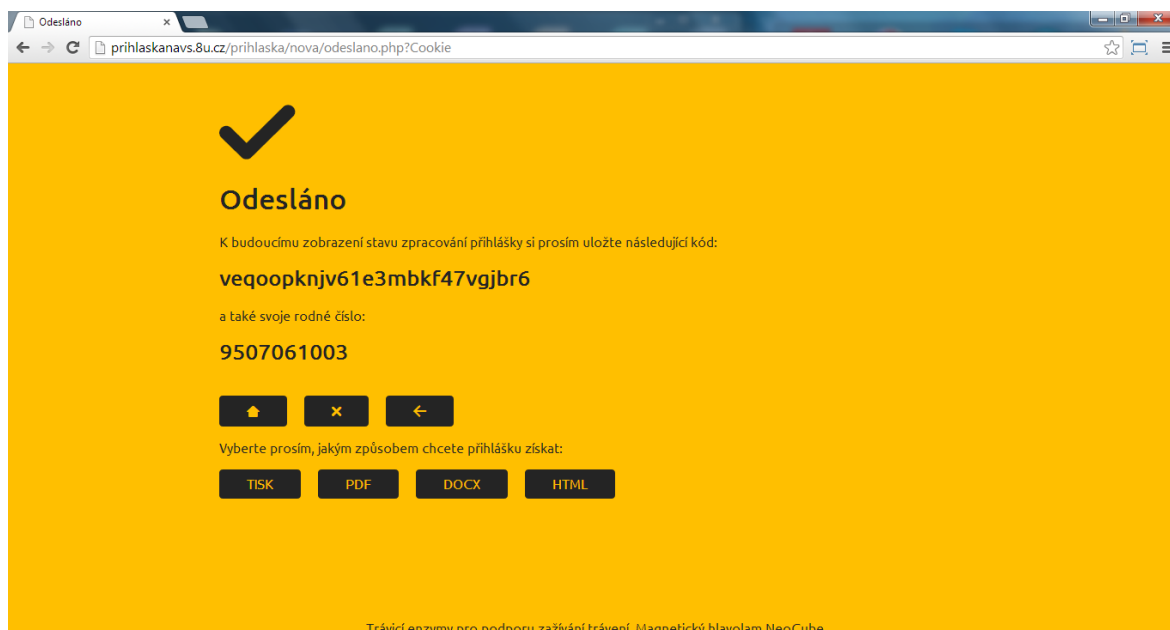
ZPĚT

DÁLE

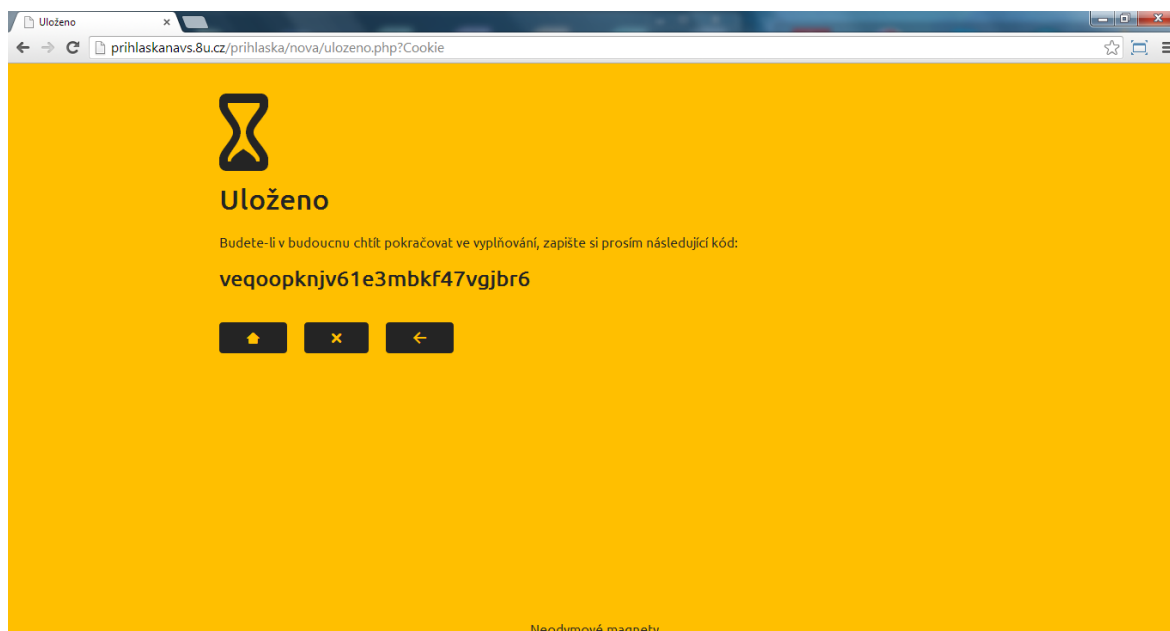
Snímek 5: Kontrola zadanych dat k odeslání přihlášky



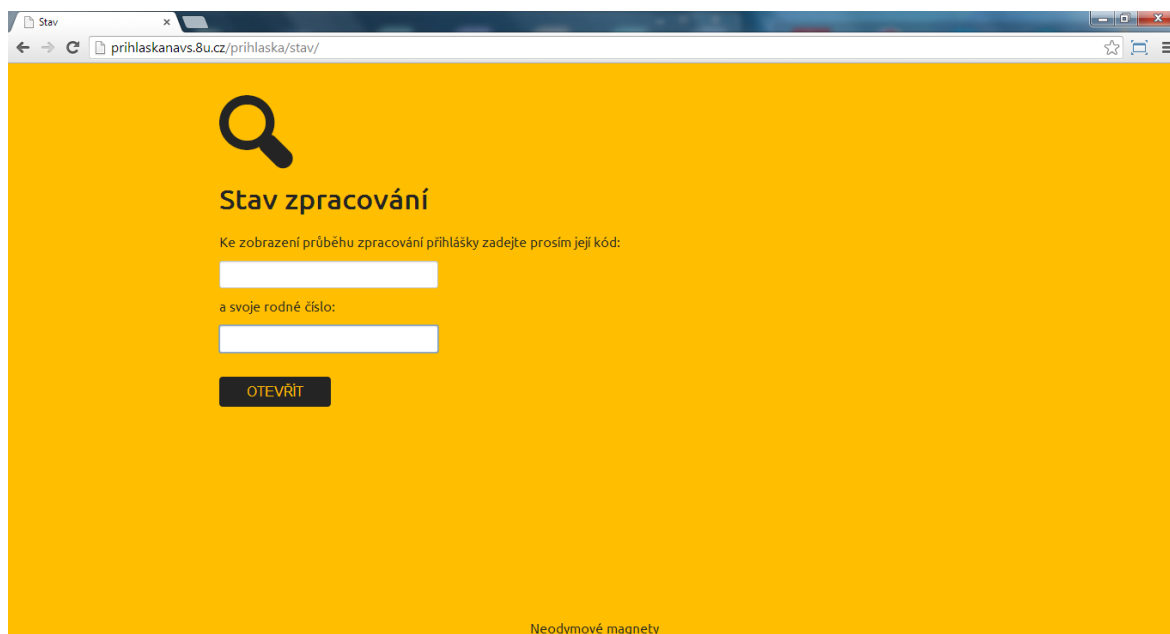
Snímek 6: Stránka informující o úspěšném odeslání přihlášky



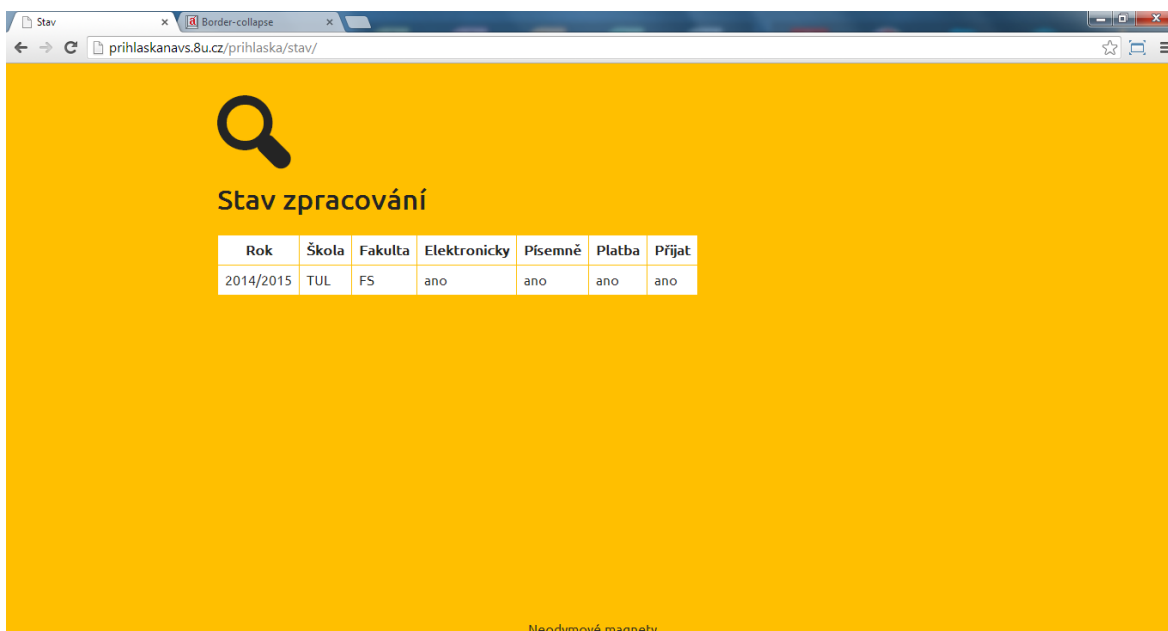
Snímek 7: Stránka informující o uložení rozpracované přihlášky do cookies prohlížeče



Snímek 8: Vstupní obrazovka pro zobrazení údajů o odeslané přihlášce



Snímek 9: Tabulka zobrazující údaje o odeslané přihlášce

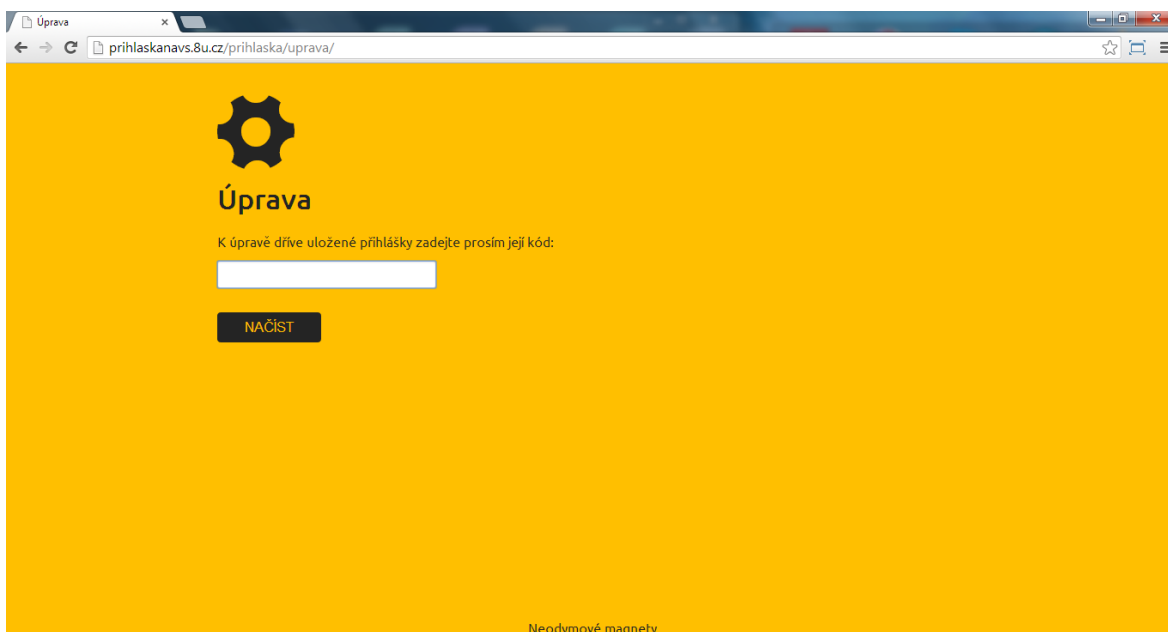


The screenshot shows a web browser window with the address `prihlaskanavs.8u.cz/prihlaska/stav/`. The page has a yellow background and a magnifying glass icon. The title is "Stav zpracování". Below it is a table with application details.

Rok	Škola	Fakulta	Elektronicky	Písemně	Platba	Přijat
2014/2015	TUL	FS	ano	ano	ano	ano

Neodymové magnety

Snímek 10: Vstupní stránka pro načtení uložené rozpracované přihlášky



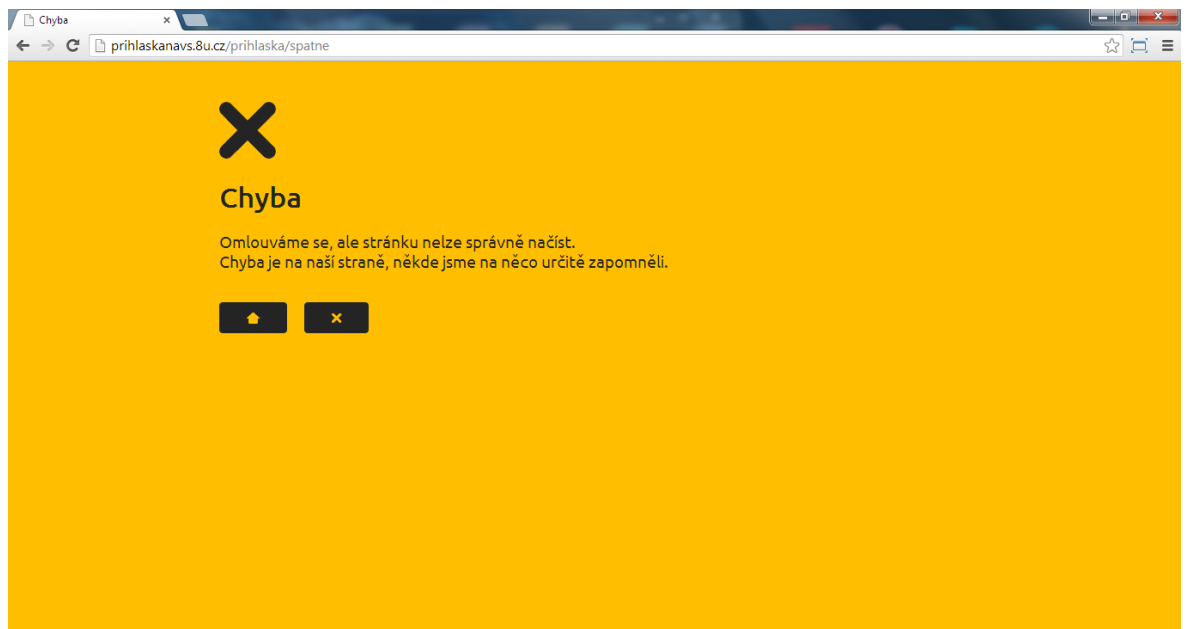
The screenshot shows a web browser window with the address `prihlaskanavs.8u.cz/prihlaska/uprava/`. The page has a yellow background and a gear icon. The title is "Úprava". Below it is a text prompt and a form field for entering a code.

K úpravě dříve uložené přihlášky zadejte prosím její kód:

NAČÍST

Neodymové magnety

Snímek 11: Chybová stránka



Snímek 12: Práce v textovém editoru Brackets s ukázkou části souboru *zpracovani.php*

