Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Framework pro vývoj matematických webových her

Plzeň 2012 Luboš Hanzík

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. května 2012

Luboš Hanzík

Abstract

This work provides a short introduction to some tools commonly used to create web applications. Namely the programming languages JavaScript and PHP as well as Adobe Flash technology are described. Another focus of this work is the matter of web security and the description of some common types of attack. The practical part begins with the description of a website called Frisky, which belongs to the Department of Mathematics. We analyzed the current implementation and identified the main issues. We propose and implement an optimal solution. Then we describe a JavaScript game called Puzzle. We analyze how it's possible to cheat in the game and then explain how we changed the code of the game to prevent that. In the next chapter we talk about another goal of this work, which was to create a simple library to save and to present results of web games. The next chapter is dedicated to another library that we created. This library provides means for communication of visitors of a web page with each other. In the last chapter, we discuss the results and propose some future research directions.

Obsah

1 Úvod

Katedra matematiky (KMA) usiluje o modernizaci a vylepšování svých webových stránek. Snahou je, aby byly používány moderní technologie jako např. HTML5 a CSS3. Zároveň je dbáno na zabezpečení stránek proti napadení.

Cílem této práce bylo podpořit rozvoj webových stránek KMA. V teoretické části této práce jsou nejprve popsány prostředky pro tvorbu webových stránek, jako jsou jazyky PHP a JavaScript (spolu s rozborem knihovny jQuery) a také je stručně popsána technologie Adobe Flash. Dále se v této části práce zabýváme oblastí bezpečnosti na webu. Kromě základních principů bezpečnosti jsou rozebrány i některé běžné způsoby napadení.

V praktické části práce čtenář nejprve nalezne analýzu nedostatků webových stránek KMA, zvaných Frisky. Následně je popsáno, jak byly identifikované problémy odstraněny. Další kapitola je věnována rozboru webové hry Puzzle. Tento rozbor je zaměřen na popis možných způsobů, jak ve hře podvádět. Následně jsou čtenáři vysvětleny kroky, které byly učiněny pro zamezení podvádění. Předposlední kapitola praktické části je věnována také webovým hrám. Zde se podíváme na programátorské rozhraní, které bylo vytvořeno pro ukládání a zveřejňování výsledků her. Poslední kapitolu praktické části považuji za nejdůležitější. V ní je rozebrána knihovna vytvořená v rámci této diplomové práce. Jedná se o knihovnu umožňující návštěvníkům webových stránek jejich zájemnou komunikaci. Jedním z poskytovaných způsobů komunikace je jednoduchý chat. Daleko zajímavější je ale komunikace prostřednictvím kreslení na HTML5 element *canvas*. Uživatelé mohou na tento element kreslit myší a knihovna zajišťuje ukládání nakreslených čar na serveru a jejich distribuci k ostatním návštěvníkům stránek.

2 JavaScript

V této kapitole se budu věnovat jazyku JavaScript. Nejvýznamější použití tohoto jazyka je ke psaní aplikací, které jsou vykonávány webovými prohlížeči. JavaScript bude významným nástrojem při tvorbě praktické části této diplomové práce. V této kapitole se podíváme na jeho vznik a na základy práce s tímto jazykem.

2.1 Historie JavaScriptu

Začátkem devadesátých let dvacátého století, kdy byl vytvořen World Wide Web, neexistovala žádná interakce s webovými stránkami. Bylo tedy potřeba vytvořit programovací jazyk, který by dovolil nadefinovat, jak má stránka reagovat na akce od uživatele, aniž by bylo třeba stránku načíst znovu ze serveru.

V této době se nejvíce používaly dva prohlížeče: Netscape Navigator a Internet Explorer. Firma Netscape jako první zavedla programovací jazyk, který přinesl interaktivitu do webových stránek. Jmenoval se Livescript a byl přímo zaintergován do jejich prohlížeče. Tehdy začal získávat na popularitě programovací jazyk Java. Firma Netscape se tedy rozhodla přejmenovat Livescript na JavaScript. Kromě podobné syntaxe ovšem nemají jazyky Java a JavaScript mnoho společného.

Aby Internet Explorer nezůstal pozadu, byl brzy také obohacen o podporu dokonce dvou programovacích jazyků. První se jmenoval vbscript a byl založen na programovacím jazyce BASIC. Druhý byl nazván JScript a velmi se podobal JavaScriptu.

Vzhledem k důležitosti JavaScriptu se jej v roce 1996 ujala standardizační organizace ECMA, která pak byla zodpovědná za další vývoj tohoto jazyka. JavaScript byl poté přejmenován na ECMAScript nebo ECMA-262. Většina lidí ale tento jazyk stále označuje jako JavaScript. Více informací o historii jazyka naleznete v [1].

2.2 Základy JavaScriptu

Programovací jazyk JavaScript je *case sensitive* (rozlišuje malá a velká písmena). Každý příkaz se ukončuje středníkem. Pokud chceme kód JavaScriptu vložit přímo do HTML souboru, umístíme jej do elementu SCRIPT:

```
<script type="text/javascript">
document.write("<h1>Hello World!</h1>");
</script>
```

Většinou je ale nejlepší psát skripty do samostatného souboru, který pak vložíme do HTML stránky takto:

```
<script src="muj_skript.js" type="text/javascript" />
```

Proměnné JavaScript je slabě typovaný jazyk. Při deklaraci proměnných tedy neuvádíme jejich datový typ. Lokální proměnné se deklarují uvnitř funkcí a před názvem proměnné se píše klíčové slovo *var*. Každá proměnná, která je deklarována vně funkce, se stává globální proměnnou. Proměnné deklarované bez použití klíčového slova *var* se také stávají globálními proměnnými.

Cykly a podmínky Syntax podmínky i cyklů je prakticky stejný s programovacími jazyky Java a C. Uvedu zde ale přesto ukázky těchto konstrukcí.

Smyčka *for*:

```
var i;
for (i=0;i<=5;i++) {
  // nějaký kód
}
```

Cyklus *while*:

```
var i=0;
while (i<=5) {
  // nějaký kód
}
```

Cyklus *for..in* je používán k procházení parametrů objektu:

```
var osoba={krestniJmeno:"Alois",prijmeni:"Manas",vek:25};
for (x in osoba) {
  document.write(osoba[x] + " ");
}
```

Podmínka *if*:

```
if (podmínka1){
  //zde umístěný kód se vykoná, pokud je podmínka1
  //splněna
}
else if (podmínka2){
```

```
  //zde umístěný kód se vykoná, pokud je podmínka2
  //splněna
}
else{
  //zde umístěný kód se vykoná, pokud
  //nejsou podmínka1 ani podmínka2 splněny
}
```

Větvení příkazem *switch*:

```
switch(n) {
case 1:
  //kód, který má být vykonán,
  //pokud proměnná n nabyde hodnoty 1
  break;
case 2:
  //kód, který má být vykonán,
  //pokud proměnná n nabyde hodnoty 2
  break;
default:
  //kód, který má být vykonán,
  //pokud n nenabývá hodnoty 1 ani 2
}
```

Funkce Funkce se deklarují klíčovým slovem *function*:

```
function nazevFunkce(parametr1,...,parametrN) {
  //kód funkce
}
```

Funkce vracejí svojí návratovou hodnotu příkazem *return*.

Další základní informace o jazyce JavaScript naleznete v [2].

2.3 Document Object Model

Aby mohl JavaScript pracovat s HTML dokumentem, potřebuje, aby prohlížeč vytvářel hierarchickou reprezentaci dokumentu, která se nazývá Document Object Model (DOM). Prvek nejvýše v DOM hierarchii je *Navigator* - reprezentuje samotný prohlížeč. Navigator poskytuje například informaci o názvu a verzi prohlížeče, zda má prohlížeč zapnuté cookies atd. Na další úrovni DOM se nachází objekt *Window*. Ten reprezentuje okno prohlížeče, lze z něj zjistit např. šířku a výšku okna, souřadnice okna na obrazovce

a další. Nejdůležitější prvek na další úrovni stromu je objekt *Document*. Reprezentuje HTML dokument a poskytuje přístup k jednotlivým HTML elementům. Obsahuje tyto kolekce: anchors, forms, images a links. Tyto kolekce lze použít k práci s příslušnými HTML elementy. Další možností, jak přistupovat k elementům, je pomocí následujících metod objektu Document: getElementById(), getElementsByName(), getElementsByTagName(). Tyto metody dovolují přístup k elementům na základě atributů *ID* a *name*, a na základě názvu elementu.

2.4 Události

Události (events) jsou akce, které mohou být detekovány JavaScriptem. Každý element webové stránky má určitou sadu událostí, na které je možné JavaScriptem reagovat. Pro nastavení reakce JavaScriptu na událost se použije příslušný atribut HTML elementu a jako jeho hodnota se nastaví název volané funkce JavaScriptu. V následujícím příkladu chceme reagovat na klepnutí na tlačítko. Použijeme proto atribut onclick. Na tuto událost se zde reaguje funkcí *mojeFunkce*.

```
<input type="button" onclick="mojeFunkce()" value="Tlačítko" />
```

Kompletní seznam událostí naleznete v tabulce 2.1.

Více informace o událostech naleznete v [3].

2.5 Knihovna jQuery

jQuery je významnou a hojně používanou knihovnou pro jazyk JavaScript. Je oblíbená díky tomu, jak významně usnadňuje programátorům práci. jQuery mění způsob, jakým se píší JavaScriptové aplikace, podporuje snadnou manipulaci s HTML elementy, modifikaci CSS, snadnou tvorbu efektů a animací. V neposlední řadě také ulehčuje používání technologie AJAX. Z těchto důvodů bude jQuery velmi významným nástrojem při realizaci praktické části této práce. Více informací o knihovně jQuery naleznete v [4].

2.5.1 Začínáme s jQuery

Aby mohly naše skripty využívat funkcí jQuery, musíme nejprve vložit odkaz na tuto knihovnu do naší HTML hlavičky, a to takto:

```
<script type="text/javascript" src="jquery.min.js"></script>
```

Pochopitelně obsah atributu *src* je potřeba změnit podle toho, jak je konkrétně pojmenován náš soubor s knihovnou jQuery.

Událost	**Popis**
onAbort	při zrušení nahrávání obrázku
onBlur	daný objekt je deaktivován
onChange	uživatel změnil obsah prvku formuláře
onClick	při klepnutí na objekt
onDblClick	při poklepání na objekt
onDragDrop	při přetáhnutí ikony do prohlížeče
onError	když nastane chyba JavaScriptu
onFocus	při aktivaci objektu
onKeyDown	když uživatel stiskne klávesu
onKeyPress	když uživatel stiskne nebo drží stisknutou klávesu
onKeyUp	když uživatel pustí klávesu
onLoad	po dokončení načítání celé stránky
onMouseDown	při stisknutí tlačítka na myši
onMouseMove	při pohnutí myší
onMouseOut	při opuštění daného objektu kurzorem myši
onMouseOver	při najetí kurzorem myši na daný objekt
onMouseUp	když uživatel pustí tlačítko myši
onMove	když uživatel pohne oknem prohlížeče
onReset	když uživatel vyčistí formulář tlačitkem reset
onResize	když uživatel změní velikost okna prohlížeče
onSelect	když uživatel označí text v textovém poli
onSubmit	když uživatel klepne na tlačítko submit daného formuláře
onUnload	když uživatel opustí stránku

Tabulka 2.1: Tabulka událostí

Funkce knihovny jQuery by se měly volat až když je dokument plně načten, proto by se měla vždy používat funkce *ready*:

```
jQuery.noConflict();
jQuery(document).ready(function() {
  //zde může být kód využívající funkcí jQuery
});
```

Ve výše uvedeném příkladu je vidět použití další důležité funkce jQuery - noConflict. Jako zástupce pro identifikátor jQuery se implicitně používá *$*. Problémem je, že identifikátor *$* používají i jiné knihovny, např. *Prototype*. Řešením těchto konfliktů je právě funkce noConflict, jejímž zavoláním se jQuery vzdá identifikátoru *$*. Identifikátor jQuery lze ovšem nadále používat. Chceme-li si nadefinovat jiný identifikátor, lze toho dosáhnout takto:

```
//$j bude novým zástupcem pro identifikátor jQuery
```

```
$j = jQuery.noConflict();
$j(document).ready(function() {
  $j("#mujobrazek").hide();
});
```

Následující ukázky jQuery budou operovat nad tímto HTML souborem:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://
    www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
    <head>
    <script type="text/javascript" src="jquery.min.js"></script>
    <script type="text/javascript" src="example.js"></script>

    <style type="text/css">
    .kotata {
      position: absolute;
    }
    #kote1 {left:0px;}
    #kote2 {left:160px;}
    #kote3 {left:320px;}
    #kote4 {left:480px;}
    </style>
    </head>

<body>
<div id="galerie" name="galerieKotat">
<img src="obrazky/kote1.jpg" width="150" height="102" class="kotata
    " id="kote1"/>
<img src="obrazky/kote2.jpg" width="150" height="102" class="kotata
    " id="kote2"/>
<img src="obrazky/kote3.jpg" width="150" height="102" class="kotata
    " id="kote3"/>
<img src="obrazky/kote4.jpg" width="150" height="102" class="kotata
    " id="kote4"/>

<div id="zanorenyDiv" name="muj div1">
  <div id="viceZanorenyDiv" name="mujDiv2">
  </div>
</div>
</div>
</body>
</html>
```

Obrázky mají absolutní polohu kvůli potřebám příkladu demonstrujícího animace.

2.5.2 Selektory

Selektory slouží k vybrání HTML elementů, se kterými chceme pracovat. Jejich syntaxe je velmi podobná syntaxi CSS. Uveďme si některé příklady:

jQuery("#kote1") vybere element s id *kote1*.

jQuery(".kotata") vybere všechny elementy patřící do třídy *kotata*.

jQuery("#galerie > div") vybere přímé potomky elementu s id *galerie*, které jsou typu DIV. V příkladu tedy vybere element s id *zanorenyDiv*.

jQuery("#galerie div") vybere všechny potomky elementu s id *galerie*, které jsou typu DIV. V příkladu tedy vybere elementy *zanorenyDiv* i *viceZanorenyDiv*.

Existují i speciální selektory, např. pro výběr pouze prvního z množiny vybraných elementů:

jQuery(".kotata:first") vybere element s id *kote1*.

Elementy lze také vybírat podle hodnot jejich atributů:

jQuery("div[name]") vybere všechny elementy DIV, které mají atribut *name*.

jQuery("div[name='galerieKotat']") vybere element DIV, jehož atribut *name* má hodnotu *galerieKotat*.

jQuery("div[name!='galerieKotat']") vybere všechny elementy DIV, jejichž atribut *name* nemá hodnotu *galerieKotat*.

jQuery("div[name ~='muj']") vybere všechny elementy DIV, jejichž atribut *name* obsahuje slovo *muj* oddělené mezerami, tedy *zanorenyDiv*.

jQuery("div[name*='muj']") vybere všechny elementy DIV, jejichž atribut *name* obsahuje řetězec *muj*, tj. *zanorenyDiv* a *viceZanorenyDiv*.

jQuery("img[src$='.jpg']") vybere všechny elementy IMG, jejichž atribut *src* končí řetězcem *.jpg*.

jQuery("div[name^='galerie']") vybere všechny elementy DIV, jejichž atribut *name* začíná řetězcem *galerie*.

2.5.3 Reakce na události

V JavaScriptu se standardně definují reakce na události použitím speciálním atributů HTML elementů, jako např. *onclick* apod. Hodnotou takového atributu pak je název funkce, která má být zavolána po nastání dané události. jQuery nám umožňuje definovat reakce na události, aniž bychom museli modifikovat HTML kód. Využijeme k tomu selektory a speciální funkce, např:

```
jQuery("img").click(function(){
  alert(jQuery(this).attr("id"));
});
```

Výše uvedený příklad reaguje na události klepnutí myší na libovolný element IMG. JavaScript pak nechá vyskočit okénko, které vypíše id elementu, na který bylo klepnuto. Stejného efektu lze dosáhnout i použitím funkce *bind*:

```
jQuery("img").bind("click", function(){
  alert(jQuery(this).attr("id"));
});
```

Kromě *click* nabízí samozřejmě jQuery i další funkce pro jiné typy událostí, jako např. *hover* (uživatel najede myší na daný element), *mouseout* (uživatel sjede myší z daného elementu) a další.

2.5.4 Animace

jQuery dovoluje provádět animace jako přechod mezi různými hodnotami CSS parametrů vybraných HTML elementů. Posledním parametrem animačních funkcí je tzv. *callback* funkce. Callback funkce se používají z toho důvodu, že kód, který následuje za příkazem zavolání animační funkce, se vykonává paralelně s danou animací. Chceme-li ovšem nějakou činnost provést až po dokončení animace, použijeme callback funkci. Ukažme si jednoduchý příklad:

```
jQuery("#kote3").fadeOut(1000, function() {
  alert("Kote 3 zmizelo");
});
```

Animace nechá zmizet element s id *kote3*, její trvání je jedna sekunda (1000 ms). Po skončení animace vyskočí okénko s nápisem Kote 3 zmizelo.

jQuery nabízí řadu speciálních animačních funkcí, podívejme se na některé příklady:

jQuery(selektor).hide(doba_trvani_animace , callback) schová všechny vybrané elementy. Po dokončení animace je CSS vlastnost *display* nastavena na hodnotu *none*.

jQuery(selektor).show(doba_trvani_animace, callback) zobrazí všechny vybrané elementy, pokud byly schované.

jQuery(selektor).fadeOut(doba_trvani_animace, callback) nechá také zmizet vybrané elementy, ale na rozdíl od funkce *hide* toho docílí plynulou změnou CSS vlastnosti *opacity* na hodnotu *0*.

jQuery(selektor).fadeIn(doba_trvani_animace, callback) opět ukáže vybrané elementy, pokud byly předtím schovány funkcí *fadeOut*.

jQuery(selektor).fadeTo(doba_trvani_animace, opacity, callback) lze použít, pokud nechceme schovat elementy úplně. Parametr *opacity* nám dovoluje nastavit výslednou hodnotu průhlednosti elementů.

jQuery(selektor).slideUp(doba_trvani_animace, callback) je další způsob, jak nechat zmizet vybrané elementy, tentokrát změnou jejich výšky.

jQuery(selektor).slideDown(doba_trvani_animace, callback) znovu ukáže elementy schované funkcí *slideUp*.

Nejdůležitější je ale funkce animate, která nám dovoluje animovat libovolné CSS vlastnosti, které má smysl animovat. Její syntax je tento:

jQuery(selektor).animate(animovane_vlastnosti, doba_trvani_animace, easing, callback). Parametr *animovane_vlastnosti* slouží k vyjmenování množiny vlastností, které mají být animovány, spolu s jejich koncovou hodnotou. Parametr *easing* slouží k výběru funkce, která udává rychlost animace v každém bodu vykonání animace. Implicitní hodnota je *swing*, druhou možností je hodnota *linear*, která říká, že animace má být provedena konstantní rychlostí.

Složitější příklad, který předvádí použití funkce *animate* a také volání callback funkce v naší uživatelské funkci, lze nalézt v příloze B.

2.5.5 AJAX

AJAX (Asynchronous JavaScript and XML) reprezentuje skupinu webových technologií, které jsou použity na straně klienta ke tvorbě asynchronních webových aplikací. S použitím AJAX je možná výměna dat mezi klientem a serverem, aniž by se tím narušovalo zobrazení nebo chování aktuální stránky. AJAX je tvořen těmito technologiemi:

- HTML (nebo XHTML) a CSS pro prezentaci dat.
- DOM (Document Object Model) pro dynamické zobrazení a interakci s daty.
- XML pro výměnu dat a XSLT pro jejich manipulaci.
- XMLHttpRequest objekt pro asynchronní komunikaci.
- JavaScript.

Data není nutné přenášet ve formátu XML. Není tedy ani povinné používat XSLT k jejich manipulaci. Alternativním formátem pro přenos dat je JSON (JavaScript Object Notation). Může být ale přenášen i obyčejný text. Další informace o AJAX naleznete v [5].

Knihovna jQuery poskytuje velmi dobrou podporu pro AJAX. Následující ukázka kódu pošle na server asynchronní HTTP dotaz:

```
jQuery.ajax({
  url: "skript.php",
  type: 'get',
  data: {"parametr1" : hodnota1}
});
```

Parametr *url* slouží k nastavení URL, na které má být HTTP dotaz zaslán. Pro nastavení typu požadavku použijeme parametr *type*. Může nabýt hodnot *get* nebo *post*. Parametry HTTP dotazu nastavíme parametrem *data*. Informace o dalších parametrech funkce *ajax* naleznete v [6].

2.5.6 Knihovna qTip 2.0

V praktické části této práce potřebuji vytvářet tooltipy k obrázkům v galerii. Standardní tooltipy prohlížeče nejsou dostačující, proto jsem se rozhodl použít knihovnu qTip 2.0. Můj zdrojový kód pro vytvoření tooltipu vypadá následovně:

```
jQuery("#img"+currid).qtip({
  id: 'qtip'+currid,
  content: {
```

```
    text:
    '<div class="bubble"><div>Uložil:
    <span class="bubble-author">
      '+getLoginFromFullname(currsender)+'
    </span>
    </div>
    <div>
      Datum:'+dateString+'</div><div> Popis: '+currdescr+'
    </div>
    </div>'
  },
  position: {
    my: 'bottom center',
    at: 'top center'
  },
  show: {
    event: 'mouseenter',
    solo: true
  },
  hide: {
    event: 'mouseleave',
    fixed: true,
    delay:200
  },
  style: {
    classes: 'ui-tooltip-dark',
    tip: {
      corner: true
    }
  }
});
```

Stručně zde vysvětlím význam jednotlivých parametrů tooltipu:

- *id* - nastavuje atribut *id* tooltipu, lze jej použít k jednoduché identifikaci tooltipu v dokumentu.
- *content* - obsahuje v sobě atribut *text*. Jeho hodnotou je potom HTML, které bude tvořit obsah tooltipu.
- *position* - slouží k nastavení pozice, kde se má tooltip zobrazovat. Zde je nastavena tak, že střed dolního okraje tooltipu má být umístěn nad prostředek horního okraje obrázku.

- *show* - vnořený atribut *event* slouží k nastavení událostí, při kterých se má tooltip objevit. Zde při najetí myší na daný obrázek. Dále atribut *solo* jsem nastavil na hodnotu *true*, což říká, že má být vždy zobrazen pouze jeden tooltip.
- *hide* - i zde je vnořen atribut *event*. Tím nastavujeme, při jakých událostech má být tooltip schován. Zde při opuštění obrázku myší. Atribut *fixed* nastavený na hodnotu *true* znamená, že tooltip nemá být schován, pokud na něj najedeme myší. Atribut *delay* nastavuje dobu v milisekundách, po které se má tooltip schovat, když obrázek nebo tooltip opustíme myší.
- *style* - slouží k nastavení vzhledu tooltipu. Vnořený atribut *classes* nám dovoluje vybrat CSS třídy, které mají být tooltipům přiřazeny. Knihovna obsahuje několik tříd, ze kterých si můžeme vybírat, nebo je možné si nadefinovat i vlastní vzhled tooltipu. Druhý vnořený atribut *tip* slouží k nastavení toho, zda se má na okraji tooltipu zobrazit malá šipka, která ukazuje na element, ke kterému tooltip přísluší.

Další informace o knihovně qTip 2.0 naleznete v [7].

3 PHP

PHP je programovací jazyk, který byl navržen pro tvorbu dynamických webových stránek. Tento jazyk bude mít velké uplatnění v praktické části této práce. V této kapitole se nejprve podíváme na vznik PHP a také na základy programování v tomto jazyce.

3.1 Historie PHP

První verzi PHP vytvořil Rasmus Lerdorf v roce 1994. Jednalo se o jednoduchou sadu nástrojů, kterou Lerdorf používal pro sledování návštěvnosti svého osobního webu. Nazval ji Personal Home Page Tools. Postupem času vznikly požadavky na širší funkcionalitu PHP Tools. Lerdorf vytvořil novou verzi, která podporovala práci s databázemi a mnoho dalšího. Díky tomu bylo možné používat tento framework k tvorbě jednoduchých dynamických webových stránek. V červnu roku 1995 zveřejnil Lerdorf zdrojový kód PHP Tools.

V září tohoto roku bylo PHP opět rozšířeno a na krátkou dobu se navíc upustilo od tohoto názvu. Nové označení bylo FI (Forms Interpreter). Programový kód se nyní vkládal přímo do HTML (do HTML komentářů). Syntax byl velmi podobný jazyku Perl, i když byl mnohem omezenější. O měsíc později byla vydána další kompletně přepsaná verze. S ní přišel i návrat k názvu PHP. Jazyk byl úmyslně navržen tak, aby byl svojí strukturou blízký jazyku C. Zatím byl ale stále omezen na UNIX systémy.

Další verze vyšla v roce 1996, jejíž označení kombinovalo předchozí názvy - PHP/FI. V této verzi se už PHP začalo skutečně přetvářet ze sady nástrojů na samostatný jazyk. Obsahovala podporu pro databáze DBM, mSQL a Postgres95, cookies a další. PHP/FI bylo prohlášeno za verzi 2.0.

V roce 1997 začali Andi Gutmans a Zeev Suraski přepisovat parser PHP/FI 2.0 pro potřeby jejich univerzitního projektu. Spolu s Lerdorfem prodiskutovali různé aspekty stávající implementace a následně začali vyvíjet nový nezávislý programovací jazyk. Tento jazyk byl nově označován pouze jako PHP, což je rekurzivní akronym s významem PHP: Hypertext Preprocessor. Nejsilnější stránkou PHP 3.0 se stala jeho rozšiřitelnost spolu s rozhraními pro mnoho databází a protokolů. Další velmi důležitou vlastností verze 3.0 bylo zavedení podpory pro objektově orientované programování. V červnu 1998 bylo PHP 3.0 prohlášeno za oficiálního nástupce PHP/FI 2.0. Aktivní vývoj PHP/FI 2.0 byl oficiálně ukončen. PHP 3.0 už bylo možné provozovat i na systémech Windows a Macintosh.

V roce 1998, krátce po vydání PHP 3.0, začali Andi Gutmans a Zeev Suraski přepisovat jádro jazyka PHP. Cílem bylo zvýšit efektivitu složitých aplikací a zlepšit modularitu kódu jazyka PHP. Na základě těchto požadavků byl vytvořen nový engine nazvaný jako Zend Engine. Na něm bylo postaveno PHP 4.0. Do této verze bylo navíc přidáno mnoho nových funkcí jako podpora pro mnoho webových serverů, HTTP sessions, bufferování výstupu, několik nových jazykových konstrukcí a další. PHP 4.0 bylo vydáno v květnu 2000.

PHP 5.0 vyšlo v červenci 2004. Používá jádro Zend Engine 2.0 s novým objektovým modelem a spoustou dalších funkcí.

Více informací o vzniku jazyka PHP naleznete v [8].

3.2 Základy PHP

Tato podkapitola obsahuje seznámení s jazykem PHP. Více informací o práci s tímto jazykem naleznete v [9].

Základní syntax PHP kód začíná řetězcem <?php a končí ?>. Na serverech s podporou zkrácené formy lze začít PHP kód řetězcem <?, ale pro lepší přenositelnost se to nedoporučuje. PHP soubory standardně obsahují HTML značky spolu s programovým kódem PHP. Výpis se provádí příkazem echo:

```
<?php
echo "Ahoj světe";
?>
```

Příklad také ukazuje, že každá řádka kódu se musí ukončit středníkem. Komentáře se zapisují podobně jako v jazyce C, řádkové začínají řetězcem // a blokové se umísťují mezi /* a */.

Proměnné Každá proměnná v PHP začíná znakem $. Proměnné jsou slabě typované. Programátor tedy nedeklaruje typ proměnné, neboť PHP určí typ automaticky podle kontextu, ve kterém je proměnná použita. Navíc se typ proměnné může automaticky měnit:

```
<?php
$a = 10; //$a je nyní typu integer (celé číslo)
$a += 0.1;  //nyní je $a float (desetinné číslo)
$a = "ahoj"; //$a je teď typu string (řetězec)
?>
```

Podmínky Podmínky jsou opět podobné klasickým programovacím jazykům jako je C nebo Java. Podmínka if vypadá takto:

```
if(podminka1) {
  //kód, který je zde, se provede, pokud je podminka1
  //splněna
}
elseif(podminka2) {
  //kód, který je zde, se provede, pokud podminka2
  //má hodnotu true a podminka1 má
  //hodnotu false
}
else {
  //kód, který je zde, se provede, pokud
  //podminka1 i podminka2 mají hodnotu false
}
```

Druhým typem větvení programu je příkaz switch:

```
switch ($promenna) {
case hodnota1:
  //provede se, pokud $promenna==hodnota1
  break;
case hodnota2:
  //provede se, pokud $promenna==hodnota2
  break;
default:
  //provede se, pokud $promenna!=hodnota1
  //a zároveň $promenna!=hodnota2
}
```

Příkaz switch umí pracovat nejen s proměnnými typu integer, ale i s typy float a string.

Cykly I syntax cyklů se velmi podobá jazyku C. Smyčka for vypadá takto:

```
$pocetIteraci = 10;
for ($i=1; $i<=$pocetIteraci; $i++) {
  echo "Iterace " . $i . "<br/>\n";
}
```

Pro pohodlné procházení polí nám PHP poskytuje smyčku foreach:

```
$pole=array("prvek1","prvek2","prvek3");
foreach ($pole as $hodnotaPrvkuPole) {
  echo $hodnotaPrvkuPole . "<br/>\n";
}
```

Posledním typem cyklu jsou smyčky while a do-while:

```
$i = 1;
while($i%5 != 0) {
  echo $i . "<br/>\n";
  $i++;
}
do {
  echo $i . "<br/>\n";
  $i++;
} while($i%5 != 0)
```

Pole K vytváření polí používáme konstrukci array(). PHP podporuje číselně indexovaná pole, ale i indexování řetězcem (asociativní pole). Následující příklad ukazuje tvorbu číselně indexovaného pole:

```
$pole = array(0=>10, 1=>20, 30);
$pole[3] = 40;
$pole[4] = 50;
$pole[] = 60;
print_r($pole);
//vypíše:
//Array ( [0] => 10 [1] => 20 [2] => 30
//[3] => 40 [4] => 50 [5] => 60 )
```

První řádka výše uvedeného příkladu ukazuje tvorbu pole, které po vytvoření obsahuje 3 prvky. Je zde vidět, že můžeme sami definovat indexy prvků, nebo lze nechat PHP indexovat prvky automaticky, což je zde případ prvku s hodnotou 30. Následující 3 řádky ukazují, že to samé platí i pro vytváření dalších prvků pole.

Další příklad ukazuje práci s asociativním polem. Dále ukazuje, že je možné kombinovat indexování pomocí řetězců i čísel. Stejně tak i hodnoty uložené v jednotlivých prvcích pole mohou být různých datových typů:

```
$pole = array("prijmeni"=>"Manas");
$pole["jmeno"] = "Alois";
$pole[] = 30;    //index bude 0
echo $pole["jmeno"]." ".$pole["prijmeni"].
  ", vek: ".$pole[0]."<br/>\n";
```

Prvek pole může být také pole, čímž lze tvořit vícedimenzionální pole.

Funkce Definice funkcí začínají klíčovým slovem function. V definici funkce se nespecifikuje návratový typ ani datové typy vstupních parametrů (i když od verze 5 lze definovat, že parametr musí být objektem specifikované třídy). Následující příklad ukazuje syntax funkcí a jejich volání:

```
function sectiDveCisla($cislo1, $cislo2) {
  return $cislo1+$cislo2;
}
echo sectiDveCisla(1, 2)."<br/>\n";
```

3.3 Prostředky pro komunikaci procesů

V realizační části této práce potřebuji zajistit meziprocesovou komunikaci. Proto v této podkapitole popíši prostředky jazyka PHP pro práci se sdílenou pamětí a semafory.

Sdílená paměť' PHP nabízí funkce k práci se sdílenou pamětí, kterou lze použít pro komunikaci procesů. Podívejme se na jednotlivé tyto funkce:

- *shm_attach* - získáme objekt, který použijeme pro přístup ke sdílené paměti. Prvním parametrem je celočíselný klíč, který slouží k identifikaci sdílené paměti. K získání klíče se obvykle používá funkce *ftok*, která klíč vytvoří na základě cesty k existujícímu souboru a na základě identifikátoru projektu. Druhým parametrem je velikost sdílené paměti. Třetím parametrem jsou přístupová práva ke sdílené paměti. Při prvním volání této funkce dojde k vytvoření sdílené paměti.
- *shm_detach* - provede odpojení sdílené paměti.
- *shm_has_var* - testuje, zda se ve sdílené paměti nachází proměnná se specifikovaným klíčem.
- *shm_get_var* - vrací proměnnou s daným klíčem.
- *shm_put_var* - uloží proměnnou identifikovanou daným klíčem do sdílené paměti.
- *shm_remove_var* - smaže proměnnou s daným klíčem ze sdílené paměti.
- *shm_remove* - smaže danou sdílenou paměť.

Semafory Pro výlučný přístup ke sdílené paměti je potřeba použít semafory. Pro práci s nimi nám PHP nabízí tyto funkce:

- *sem_get* - slouží k získání semaforu na základě klíče. Klíč je celé číslo, které si musíme zvolit. Je možné nastavit, kolik procesů může zároveň získat zámek. Implicitně je semafor binární. Dále je možné nastavit přístupová práva k semaforu a také zda má být zámek automaticky uvolněn po skončení skriptu.
- *sem_acquire* - získání zámku.

- *sem_release* - uvolnění zámku.
- *sem_remove* - odstraní daný semafor.

4 Adobe Flash

Adobe Flash je velmi významný nástroj pro tvorbu interaktivních animací, videí i her. V této kapitole se podíváme na historii vzniku tohoto nástroje a na některé jeho vlastnosti.

4.1 Historie

Flash se vyvinul z programu SmartSketch, jehož autorem byl Jonathan Gay. Jonathan Gay založil v roce 1993 firmu FutureWave Software, která začala pracovat na programu SmartSketch, jehož cílem bylo usnadnit kreslení perem na dotykové displeje. Ovšem v roce 1994 byla firma Go, která vyvíjela operační systém pro počítače s dotykovým displejem, koupena firmou AT&T a její činnost byla zastavena. Tím zmizel trh pro SmartSketch. FutureWave se tedy rozhodli portovat SmartSketch na Windows a Macintosh.

V roce 1995 se už začalo zdát, že by se mohl Internet stát populární záležitostí. Ve FutureWave začali přemýšlet o tom, že lidé jistě budou chtít přes internet přenášet grafiku a animace. Začali tedy přidávat animaci do SmartSketch. V té době jedinou možností, jak přehrát animaci v prohlížeči, bylo pomocí Javy. FutureWave tedy napsali jednoduchý přehrávač založený na Javě. Problémem ale bylo, že běžel neúnosně pomalu. Naštěstí na podzim roku 1995 vydali Netscape API pro tvorbu zásuvných modulů do jejich webového prohlížeče. To konečně dovolilo FutureWave vytvořit efektivně fungující přehrávač. Výsledný produkt byl pojmenován FutureSplash Animator a byl vydán v létě 1996.

V roce 1996 se FutureSplash zalíbil Microsoftu. Ti právě pracovali na jejich službě MSN (The Microsoft Network) a rozhodli se FutureSplash použít v jejich produktu. Druhým významným zákazníkem FutureWave byl Disney Online. Ti použili FutureSplash k tvorbě uživatelského rozhraní jejich webu Disney Daily Blast.

V prosinci roku 1996 koupila FutureWave Software firma Macromedia. FutureSplash Animator byl pak přejmenován na Macromedia Flash 1.0. V současnosti vývíjí a distribuuje Flash firma Adobe Systems, která v roce 2005 koupila firmu Macromedia.

Více o vzniku Adobe Flash si můžete přečíst v [10].

4.2 Vlastnosti

Flash tedy slouží ke tvorbě animací a jejich prezentaci především na webu. Umí pracovat s vektorovou i rasterovou grafikou. Umožňuje animovat text, kresby a obrázky. Podporuje řadu video a audio formátů jako např. MP3, avi nebo wmv. Flash animace mohou být navíc interaktivní. K tomu se používá skriptovací jazyk ActionScript, který dovoluje definovat reakce na různé akce, jako je např. stisknutí tlačítka, nebo může být akce svázána se snímkem animace. Další důležitou vlastností je, že animace mohou být streamovány a jejich přehrávání může začít dříve, než je celá animace stažena do přehrávače.

Flash ukládá svoje animace do souboru ve formátu SWF (Small Web Format). Obvykle mají příponu .swf a lze je přehrát ve Flash přehrávači. Případně mohou být ve formě samostatně spustitelného souboru (soubor má potom pod OS Windows příponu .exe). Dalším formátem je Flash Video. Tyto soubory mají příponu .flv. Mohou být vloženy do .swf souborů nebo může být pro jejich přehrání použit některý z přehrávačů, který má pro tento formát podporu.

Použití vektorové grafiky spolu s programovým kódem dovoluje Flash souborům, aby byly menší, než korespondující videa v jiných formátech. Na druhou stranu může přehrávání Flash souborů více zatěžovat CPU, pokud v nich obsažený programový kód provádí složité operace.

Další informace o Adobe Flash naleznete v [11].

5 Bezpečnost na webu

V této kapitole se budeme zabývat různými aspekty bezpečnosti na webu. Podíváme se také na některé známé způsoby napadení a u některých uvedu způsob obrany. Více informací na téma bezpečnosti naleznete v [12].

5.1 Cíle bezpečnosti

Bezpečnost se skládá z různých aspektů. Abychom dosáhli určité rozumné úrovně bezpečnosti, musí být bezpečnost celistvá. Dosáhnout absolutní bezpečnosti je obvykle nemožné, snahou je ale vždy riziko prolomení ochrany co nejvíce minimalizovat. Uvažuje se tak, že bezpečnost je třeba mít na takové úrovni, aby se snaha o její prolomení útočníkovi nevyplatila. O systému se tvrdí, že je bezpečný tak, jak je zabezpečený jeho nejslabší prvek. Podívejme se dále na jednotlivé koncepty bezpečnosti.

5.1.1 Autentizace

Jedná se o problém zjištění identity, typicky osoby nebo stroje. Např. před tím, než určitý systém vydá nějaké citlivé informace, musí se ujistit, že ví, s kým komunikuje.

Prvním způsobem, jak se o tom ujistit, je na základě něčeho, co komunikující osoba zná a co by neměl znát nikdo jiný. Typicky se jedná o zadání hesla. Výhodou hesel je jejich snadná implementace (na rozdíl např. od snímání otisků prstů) a obvykle ani příliš nekomplikují život uživatelům. Problémem ovšem může být, když je uživatelům dovoleno vybrat si libovolné heslo. Často se pak najde nezanedbatelné procento uživatelů, jejichž heslo je snadno uhádnutelné. Jednou možností, jak tento problém vyřešit, je, když systém sám přidělí každému uživateli heslo. Taková generovaná hesla ale jsou těžká na zapamatování. Lepší řešení je nechat uživatele vybrat si heslo, ale zároveň vynutit určitou složitost hesla - např. každé heslo musí být dlouhé alespoň osm znaků a musí obsahovat kromě písmen abecedy alespoň dvě číslice.

Druhým způsobem, jak určit identitu, je na základě něčeho, co osoba vlastní. Příkladem mohou být čipové karty. Ty mohou fungovat tak, že uživatel vloží kartu do čtečky. Karta vyšle čtečce výzvu. Uživatel musí následně zadat do čtečky PIN. Na základě PIN pak čtečka spočte odpověď, kterou předá čipové kartě. Pokud je odpověď správná, je uživatel považován za autentizovaného a přístup k informacím uloženým na kartě (případně v systému) je povolen.

Třetí způsob autentizace je na základě biometrických parametrů člověka. Jedná se například o snímání otisků prstů nebo ještě lépe o snímání obrysu ruky, kdy se zároveň také snímají otisky všech prstů. Další metody jsou například skenování duhovky nebo snímání podpisu. Každá z těchto metod vygeneruje binární reprezentaci osoby, která by měla být unikátní pro každou osobu. Při každé autentizaci uživatele se pak výsledky měření porovnávají se záznamy v databázi a zjišťuje se, zda došlo k dostatečně velké shodě, abychom mohli prohlásit, že se podařilo uživatele identifikovat. Důležitým cílem při vývoji těchto metod je, aby nedocházelo k případům, kdy biometrická metoda přidělí uživateli nesprávnou indentitu - např. prohlásí o osobě, která vůbec v databázi není, že se jedná o člověka, který má plný přístup do systému. Opačným problémem je, když není rozpoznán legitimní uživatel, jehož biometrické údaje se v databázi nacházejí.

Jako poslední poznámku o autentizaci bych rád zmínil, že pro zlepšení bezpečnosti lze výše uvedené metody identifikace kombinovat. Např. trezor banky se otevře pouze tehdy, když její ředitel si nejprve nechá oskenovat duhovku oka a pak ještě vloží svoje heslo.

5.1.2 Autorizace

Cílem autorizace je určit, jaká oprávnění má daný uživatel. V operačních systémech se toto řeší např. pomocí Access Control List (ACL). Jedná se o seznam, ve kterém je nadefinováno, ke kterým zdrojům mají jednotliví uživatelé přístup, a jaké operace (čtení, zápis, spuštění) mohou nad zdroji provádět. V lepším systému je možné zároveň uživatele přiřazovat do skupin. Každá skupina má pak opět nadefinované zdroje, ke kterým má přístup, a operace nad nimi.

Další autorizační systém je např. Bell-LaPadula model, který je určen především pro armádu. Dokumenty v systému mají nadefinovaný stupeň utajení. Každý uživatel má přiřazen stupeň oprávnění, který odpovídá některé z úrovní utajení. Prvním pravidlem, které platí v tomto systému, je No Read Up. To znamená, že uživatel nesmí číst dokumenty s vyšší úrovní utajení, než je jeho oprávnění. Druhým pravidlem je No Write Down. To znamená, že uživatel nesmí vytvořit dokument, který má nižší úroveň utajení, než je stupeň oprávnění autora. Filozofií pro toto pravidlo je, že osoby s vysokou úrovní oprávnění budou typicky generovat dokumenty, které budou obsahovat nějaké tajné informace. Proto systém zajišťuje, aby nebylo možné nedopatřením uživatele zveřejnit tajné informace.

5.1.3 Soukromí

Při komunikaci dvou stanic přes počítačovou síť je často žádoucí, aby nikdo další nemohl tuto komunikaci číst. Například v síti ethernet, kde je použit ethernetový *hub*,

je veškerá komunikace rozesílána všem stanicím. Operační systém sice standardně komunikaci, která mu není určena, zahazuje. Je ovšem možné použít tzv. promiskuitní režim, který dovolí programu číst všechny zprávy.

Pro zajištění soukromí je potřeba použít šifrování zpráv. Obvykle se šifruje na základě klíče. Klíč musí být informace, kterou znají pouze ty stanice, které chtějí spolu komunikovat. Existují algoritmy pro bezpečnou výměnu klíčů - např. algoritmus Diffie–Hellman.

5.1.4 Integrita dat

Další problém, který může při komunikaci nastat, je útok typu *man in the middle*. V situaci, kdy dvě stanice chtějí mezi sebou komunikovat, je možné, aby třetí stanice zachytávala a modifikovala posílané zprávy. V síťových protokolech se k ověření integrity dat používá například CRC (cyclic redundancy check). S použitím CRC dokážeme detekovat, že některé bity zprávy byly ztraceny nebo změněny. CRC je funkcí posílané zprávy. Tento způsob kontroly integrity je ale vhodný pouze v případě, kdy předpokládáme, že zpráva může být změněna chybou přenosu. Jako obrana proti útoku typu *man in the middle* není dostačující, neboť útočník může změnit nejen obsah zprávy, ale i CRC tak, aby příjemce modifikaci nedetekoval.

Řešením může být posílat spolu se zprávou MAC (message authentication code). MAC je funkcí nejenom samotné zprávy, ale také tajného klíče. Útočník, který nezná tajný klíč, není schopen k modifikované zprávě vygenerovat odpovídající MAC.

5.1.5 Odpovědnost

Odpovědnost je další důležitý prvek při zajištění bezpečnosti. Jedná se o to, že pokud dojde k určitému útoku, chceme být schopni zjistit, kdo útok provedl. Například pokud určitá vysoce postavená osoba ve firmě má pravomoc manipulovat s bankovním účtem firmy, musí být možno zjistit, zda si tato osoba nepřevádí peníze například na svůj osobní účet. K tomu je potřeba používat logování. Do logů se musí ukládát kdo, kdy, co provedl. Navíc musí být zajištěno, že zmiňovaný zaměstnanec firmy nemůže logy modifikovat a tím po sobě zahladit stopy.

5.1.6 Dostupnost

Dostupnost znamená, že server dokáže dostatečně rychle odpovídat na dotazy. K narušení dostupnosti se používají tzv. Denial of Service (DoS) útoky, které obvykle fungují tak, že nějaký stroj začne posílat na server nadměrný počet dotazů, čímž způsobí značné zpomalení serveru nebo i jeho pád. Situace, kdy několik stanic najedou začne

takto zahlcovat server, se nazývá Distributed Denial of Service (DDoS). K provedení takovýchto útoků se často používají počítačové viry, které, jakmile dostanou povel, začnou z postižených stanic zahlcovat určitý server dotazy. DoS útok je také např. možné provést zaplněním pevného disku postiženého stroje. Může k tomu dojít na serverech, které dovolují uživatelům ukládat soubory na disk serveru. Proto je nutné vždy limitovat množství dat, která mohou uživatelé nahrát na server.

5.1.7 Nepopiratelnost

Cílem je, aby po provedení určité transakce nemohla žádná ze zúčastněných stran tuto transakci popřít. Příkladem může být, když Alenka má zaplatit peníze Bobovi, ale nechce mu je poslat přímo, aby Bob nemohl později tuto platbu popřít. K vyřešení této situace se většinou použije třetí strana, které věří Bob i Alenka. Potom Alenka pošle peníze této třetí straně a ta oznámí Bobovi, že může Alence poslat zakoupený produkt. Jakmile Alenka obdrží objednaný produkt, dostane Bob peníze od důvěryhodné třetí strany.

5.2 Typy útoků

Webový vandalismus Jedná se o útok, jehož cílem je nahradit určité webové stránky nějakými jinými. Obětí těchto útoků mohou být typicky politické strany. Útočníkem je obvykle osoba nebo skupina osob, kterým se například nelíbí ideály a názory dané politické strany. Cílem útoku pak může být např. zesměšnit danou politickou stranu. Například v letech 1999 a 2001 došlo k útokům tohoto typu na webové stránky Bílého domu. Údajně za nimi stála skupina anti-NATO aktivistů.

Infiltrace Infiltrace je útok, kdy neautorizovaná osoba získá kontrolu nad počítačovým systémem. Tento útok lze obvykle vykonat zneužitím chyb v softwaru, jako je např. přetečení bufferu. Infitraci je možné využít ke změně stránek nějaké organizace. Horší situace může nastat, pokud útočník dokáže číst obsah databáze. Může pak např. získat z databáze tajné informace a/nebo jména a hesla uživatelů. Proto je důležité nikdy neukládat hesla v čisté textové podobě.

Phishing Phishing je metodou jak získat přihlašovací údaje uživatelů např. nějakého bankovního systém. Obvykle se realizuje pomocí e-mailu, ve kterém je uživatel informován o tom, že došlo k nějakému problému nebo jiné události, a je třeba, aby se uživatel přihlásil na stránky banky, kde se dozví bližší informace. E-mail pak obsahuje odkaz, jehož textem je správné URL banky. Ovšem parametr odkazu HREF obsahuje adresu stránky útočníka, která se snaží vypadat stejně jako legitimní stránky banky. Pokud

uživatel do formuláře na narafičené stránce zadá svoje přihlašovací údaje, zmocní se jich útočník.

Pharming Cílem této metody je, stejně jako v případě Phishingu, přelstít uživatele, aby zadal svoje přihlašovací údaje do narafičené stránky. Pharming je ale na rozdíl od Phishingu sofistikovanější. Jeho klíčem je totiž napadení DNS serveru a modifikace jeho tabulek pro převod doménového jména na IP adresu. Pokud se takovéto napadení DNS serveru podaří, nemá uživatel prakticky ani možnost si útoku všimnout. Přestože zadá do webového prohlížeče správné URL, je směrován na server útočníka.

Útoky zevnitř Často se stává, že za krádeže dat firmy mohou samotní pracovníci firmy. To je často v důsledku toho, že někteří pracovníci mají zbytečně velké pravomoce a přístup k datům, která nepotřebují ke své práce. To dává možnost úniku dat, které pak mohou být prodány jedincům nebo organizacím, které je mohou zneužít. Proto je potřeba nastavit pravomoce tak, aby se k citlivým datům dostala pouze malá množina důvěryhodných osob.

Denial of Service Tento typ útoku byl již zmiňován výše. DoS může určitým organizacím způsobit značné škody, např. pokud se jedná o internetové obchody. Ty mohou při takových útocích přijít k citelným škodám, protože legitimní uživatelé nemohou nakupovat.

5.3 Způsoby napadení počítače

5.3.1 Červi

Červ je škodlivý software, který se šíří pomocí počítačové sítě. Internet dovoluje červům šířit se velmi vysokou rychlostí. Červi obvykle zneužívají chyb v softwaru, jako je např. přetečení bufferu. Důležitou ochranou počítače je tedy mít zapnuto co nejméně služeb, které je možné z internetu napadnout. Pokud určitý operační systém v základním nastavení spouští nějakou špatně zabezpečenou aplikaci, mohou toho červi využít k velmi rychlému šíření.

Např. červ Code Red využil chyby v serveru Internet Information Server (IIS) na operačních systémech Microsoft Windows. Code Red uměl zneužít přetečení bufferu v tomto serveru. Červ se šířil velmi rychle, neboť jakmile infikoval počítač, začal neustále generovat nové IP adresy a pokoušel se napadnout další počítače na těchto adresách. Navíc antiviry jej nebyly schopny detekovat, protože neukládal nic na disk, ale byl uložen pouze v operační paměti. Bylo tedy možné se jej zbavit prostým re-

startováním počítače, ovšem pravděpodobnost, že bude počítač znovu napaden, byla vysoká.

Daleko nebezpečnější byl červ Nimda. Nimda se dokázal šířit z jednoho webového serveru na druhý. Navíc se dokázal šířit i na klienty, neboť napadal soubory na serveru. Z klientů se pak šířil dál rozesíláním e-mailů, které obsahovaly kód červa.

5.3.2 Další typy škodlivého softwaru

Rootkits Jedná se o podvodnou sadu nástrojů operačního systému. Cílem útočníka je nahradit standardní verzi nástrojů operačního systému jejich podvodnou verzí. Pokud se toto podaří, může rootkit sloužit k maskování běhu jiného škodlivého SW.

Botnets Jedná se o množinu počítačů připojených k internetu, které napadl malware. Tento typ malware je typicky neaktivní do doby, než obdrží přes internet příkaz k určité akci. Typicky se používá k DDoS útokům, kdy útočník rozešle všem strojům v botnetu příkaz, aby začali určitou IP adresu zaplavovat síťovými pakety. Dalším příkladem použití botnetu je rozesílání nevyžádané pošty.

Spyware Spyware je software, který monitoruje aktivitu uživatele počítače bez jeho svolení. Např. může monitorovat, jaké webové stránky uživatel navštěvuje a zjištěné informace posílat na centrální server k jejich dalšímu zpracování. Spyware může na základě zjištěných údajů zobrazovat uživateli infikovaného počítače reklamu.

Keyloggers Keylogger monitoruje vstupy z klávesnice a myši, což dovoluje zjistit a zcizit velmi citlivé údaje, jako jsou přihlašovací jména a hesla do různých systémů.

Adware Cílem adware je zobrazovat reklamu na obrazovce počítače. Tento typ malware tedy nemusí nutně způsobovat škodu počítači, ale pochopitelně může značně obtěžovat uživatele vyskakováním oken a podobně.

Trojské koně Jedná se o program, který se tváří jako legitimní software, ale kromě jisté žádoucí činnosti provádí tajně jinou, škodlivou činnost. Tato škodlivá činnost může odpovídat aktivitám některého z výše popsaných typů malware. Trojské koně mohou také dát útočníkovi vzdálený přístup k napadenému počítači.

5.3.3 Problém přetečení bufferu

Tento problém se typicky týká kompilovaných programovacích jazyků, jako je C nebo C++. Přetečení bufferu může například nastat při načítání řetězce ze vstupu, kdy není

omezeno, kolik znaků může uživatel zadat. Ovšem velikost bufferu, do kterého vstupní řetězeč ukládáme, omezená je. Vstup od většiny legitimních uživatelů se sice do bufferu vejde, ale pokud útočník odhalí možnost přetečení bufferu, může ji zneužít např. k nestandardnímu ukončení běhu programu nebo dokonce k řízení běhu programu. Pokud je totiž uložen buffer v zásobníku, je možné jeho přetečením přepsat návratovou adresu z podprogramu. Pokud k přepsání návratové adresy dojde omylem, způsobí to pravděpodobně nestandardní ukončení programu. Útočník ale může zvolit vstupní řetězec tak, aby přepsal návratovou adresu adresou funkce, kterou chce zavolat (a jejíž volání je normálně podmíněno například zadáním správného hesla). Pokud má útočník zdrojový kód programu, může si její adresu nechat vypsat programem. Pokud má pouze binární verzi programu, může adresu zjistit pomocí debuggeru. Útočník může dokonce vložit tímto způsobem do programu svůj vlastní kód. Tento kód se vloží za návratovou adresu do útočníkova vstupního řetězce. Návratová adresa se nastaví na adresu vloženého kódu.

Přetečení bufferu je tedy třeba předcházet kontrolováním počtu přečtených znaků. Existují i různé speciální metody, které se snaží detekovat přetečení bufferu. Jednou takovou metodou je StackGuard. Ten vkládá do zásobníku před návratovou adresu několik náhodně generovaných bytů. Pak vždy, než je proveden návrat z podprogramu, je zkontrolováno, zda ochranné byty nebyly přepsány. Pokud ano, je výpočet programu zastaven.

5.3.4 SQL Injection

SQL Injection je speciálním případem Command Injection - situace, kdy uživatel může pomocí svého vstupu vložit do programu nějaký příkaz. Obvykle k tomu může dojít v důsledku nedostatečné kontroly vstupního řetězce. Command Injection může značným způsobem narušit bezpečnost.

SQL Injection je případ, kdy uživatel může šikovným vstupním řetězcem modifikovat SQL dotaz a získat např. citlivá data z databáze. Tento problém se typicky řeší ve webových aplikacích. SQL Injection lze vyrobit, když aplikace vezme řetězec, který uživatel zadal do webového formuláře, a použije jej přímo ve svém SQL dotazu.

Problém si ilustrujme na příkladu půjčovny DVD, jejíž webová aplikace dovoluje uživatelům prohlížet si přes web historii filmů, které si zapůjčili. Každý uživatel by měl být schopen zobrazit si pouze svojí vlastní historii výpůjček. Dejme tomu, že webová aplikace chce nejprve od uživatele, aby do textového políčka vyplnil rok, za který chce historii zobrazit. Problémy mohou velmi snadno nastat, pokud SQL příkaz pro zjištění historie výpůjček bude vypadat následovně a za předpokladu, že proměnná rok je

řetězec, který zadal uživatel do políčka formuláře:

```
String sql =
"SELECT id_uzivatele, nazev,
datum_vypujcky, rok_vypujcky FROM vypujcky
WHERE id_uzivatele="+id_uzivatele+"
AND rok_vypujcky="+rok;
```

Pokud uživatel zadá číslo roku, proběhne vše v pořádku. Zadá-li ovšem uživatel následující řetězec, získá informace o výpůjčkách všech uživatelů: 2011 OR 1=1. Výsledný SQL příkaz totiž bude vypadat takto:

```
SELECT id_uzivatele, nazev,
datum_vypujcky, rok_vypujcky FROM vypujcky
WHERE id_uzivatele=1
AND rok_vypujcky=2011 OR 1=1
```

Podmínka se vyhodnocuje takto: (id_uzivatele=1 AND rok_vypujcky=2011) OR 1=1. Část podmínky za závorkou je vždy pravdivá, vypíše tedy program všechny položky v databázi.

Situace ale může být ještě mnohem horší. Řekněme, že půjčovna DVD má ve své databázi také údaje o zákaznících a to včetně jejich dat narození a čísel kreditních karet. Uživatel díky naší naivně naprogramované webové aplikaci může získat všechny tyto informace, pokud použije množinové sjednocení (UNION) a druhý příkaz SELECT:

```
2011 AND 1=0
UNION SELECT id, jmeno, datum_narozeni, cislo_platebni_karty
FROM uzivatel
```

Pokud uživatel zadá výše uvedený řetězec do formuláře, zobrazí se mu všechny citlivé informace o zákaznících půjčovny.

Escapování je jeden ze způsobů, jak upravit vstupní řetězec, aby znaky se speciálním významem byly považovány databází za obyčejná data. Např. apostrof může být databází považován za začátek nebo konec řetězce. Abychom jej mohli použít jako součást dat, musíme před něj vložit ještě jeden apostrof. Jazyk PHP poskytuje např. pro práci s databází MySQL funkci *mysql_real_escape_string*, která ve vstupním řetězci escapuje tyto znaky: \x00, \n, \r, \, ', " a \x1a.

Escapování pomáhá zabránit SQL injection pouze v případě, kdy vstupní data od uživatele vkládáme do SQL dotazu jako datový typ řetězec. K vyřešení výše uvedeného problému, kde má uživatel zadat rok, za který chce zobrazit historii výpůjček DVD, nám escapování nepomůže. V tomto případě totiž uživatel nemusí zadat žádné speciální znaky, aby mohl provést svůj útok.

Nejlepším způsobem, jak zabránit SQL injection, je použití tzv. *Prepared Statements*. Ty dovolují oddělit příkazy a data jednoznačně od sebe. Kód v jazyce Java zabezpečený proti SQL Injection by vypadal takto:

```
PreparedStatement ps = con.prepareStatement("
SELECT id_uzivatele, nazev, datum_vypujcky,
 rok_vypujcky
FROM vypujcky
WHERE id_uzivatele=? AND rok_vypujcky=?");
ps.setInt(1, id_uzivatele);
ps.setInt(2, Integer.parseInt(rok));

ResultSet rs = ps.executeQuery();
```

Otazníky v SQL příkazu nám drží místo pro hodnoty parametrů, které se pak nastavují pomocí setX metod (např. zde setInt). Tato implementace nám zaručuje, že data nebudou nikdy interpretována jako příkazy.

6 Webové stránky Frisky

Frisky patří mezi webové stránky Katedry matematiky (KMA). Účelem těchto stránek je ale spíše pobavení návštěvníků, než přímo výuka matematiky. Frisky je web plný různých javascriptových her a hraček, které mají větší či menší spojitost s matematikou nebo s KMA. Jednotlivé stránky jsou tedy velmi různorodé. Většina stránek je napsána v jazyce HTML4, některé ale využívají možností HTML5 a CSS3. Serverová část je napsána v jazyce PHP5.

6.1 Problémy k řešení

- Layout stránek není jednotný a neobsahuje menu. Vzhled také není konzistentní se stránkami Trial[1].
- Není implementováno přihlašování uživatelů.
- Stránky nejsou samostatné (nefunkčnost při přesunu na jiný server).

6.2 Analýza problémů a návrh řešení

6.2.1 Nejednotný layout stránek

Jedním z hlavních problémů webových stránek Frisky je neexistence jednotného layoutu stránek. Jediným společným prvkem stránek je hlavička. Její vzhled ale není konzistentní s novým vzhledem webových stránek Trial. Hlavička na Frisky je realizovaná tak, že se roztahuje na šířku okna prohlížeče. Obsah stránek je velmi různorodý, na některých stránkách má obsah pevnou šířku, na jiných se roztahuje přes celé okno prohlížeče.

Jako řešení jsem navrhl odstranit hlavičku a vytvořit jednotný layout stránek, který bude mít tyto základní prvky:

- Hlavička stránky, která bude obsahovat logo a případně odkazy na další stránky Katedry matematiky.
- Horní lišta obsahující prvky pro přihlašování uživatelů.
- Menu.

[1] Trial jsou další webové stránky patřící KMA.

- Obsah stránky.
- Patička.

Dále jsem navrhl, že většina stránek by měla mít jednotnou šířku, konkrétně 1020px, což odpovídá šířce menu. Mělo by být zároveň možné layout jednoduše rozšířit u stránek, jejichž obsah nelze vměstnat do této šířky.

Na stránkách Frisky také existují některé stránky, u kterých je potřeba, aby se šířka jejich obsahu měnila podle šířky okna prohlížeče. Proto jsem navrhl vytvořit speciální layout, který bude obsahovat hlavičku a obsah, jehož šířka se bude měnit podle šířky zbytku okna prohlížeče.

Menu je v současnosti na Frisky umístěno pouze na úvodní stránce, což není vyhovující. Zadavatel si ale také nepřeje, aby bylo menu umístěno napevno do všech stránek, neboť by zabíralo příliš mnoho místa. Proto bylo rozhodnuto, že při vstupu na každou stránku (s výjimkou úvodní stránky) bude menu schované a ukáže se až po stisknutí nějakého tlačítka. Umístění tohoto přepínače bylo navrženo zadavatelem poměrně nekonvenčně - menu se ukáže, když uživatel klepne na logo stránky. Na první pohled se toto může zdát jako nedobré řešení, neboť podle konvence by klepnutí na logo stránek mělo vždy vést na úvodní stránku. V případě stránek Frisky ale je takové řešení ve skutečnosti vhodné, neboť jediné, co úvodní stránka obsahuje, je menu. Nyní tedy místo toho, aby se uživatel musel vracet kvůli menu na úvodní stránku, stačí když klepne na logo a menu se mu objeví na aktuální stránce, aniž by ji musel opustit.

Dále je potřeba, aby vzhled stránek Frisky byl konzistentní se vzhledem stránek Trial. Zadavatel rozhodl, že dostatečné bude použití stejného loga a stejného obrázku v patičce.

Požadavkem zadavatele bylo, že logo musí být na každé stránce vždy v levém horním rohu okna prohlížeče, stejně jako na stránkách Trial. Dále se hlavička při skorolování stránky doprava musí schovat pod obsah stránky a při odskrolování doleva musí být hlavička vidět v levé části okna prohlížeče.

6.2.2 Neexistence přihlašování a registrace na Frisky

V současnosti na stránkách Frisky neexistuje možnost registrace ani přihlašování uživatelů. Zadavatel proto požaduje, aby bylo toto implementováno. Přihlašování by mělo být možné pomocí systému Orion i pomocí místní registrace. Místní registrace bude určena především pro uživatele, kteří nejsou registrováni v systému Orion. Dále bylo požadavkem, aby registrace na Frisky obsahovala ochranu proti registraci robotů,

musí být tedy použita CAPTCHA[2]. Zadavatel si také přál, aby uživatelé museli povinně při registraci vyplnit svojí e-mailovou adresu. Pravost této adresy by měla být ověřena aktivačním e-mailem obsahujícím URL, po jehož navštívení je uživateli dovoleno se přihlásit.

Registrace uživatelů také musí být ošetřená proti SQL a HTML injection.

6.2.3 Stránky nejsou samostatné

Jedním z požadavků zadavatele byla také samostatnost stránek. Tím je myšleno, že by mělo být možné jednotlivé hry, které se na stránkách Frisky vyskytují, překopírovat na jiný server, kde by pak okamžitě fungovaly bez potřeby jakýchkoliv úprav. Každá stránka na Frisky má svůj vlastní adresář. Je tedy potřeba, aby každá stránka měla ve svém adresáři všechny potřebné knihovny. Dále bude potřeba vytvořit novou verzi hlavního PHP skriptu. Ta by neměla obsahovat žadné prvky layoutu stránky s výjimkou samotného obsahu.

6.3 Řešení jednotlivých problémů

Podívejme se, jak byly vyřešeny jednotlivé problémy na základě předchozí analýzy.

6.3.1 Nejednotný layout stránek

Podle předchozí analýzy jsem vytvořil layout stránek, který obsahuje všechny výše uvedené prvky. Základ stránky (horní lišta, obsah, patička) je složen z DIV HTML elementů, které mají CSS atribut *position* nastaven na hodnotu *relative* a parametr *float* nastaven na *left*. Tento přístup je považován za lepší než tvorba layoutu pomocí HTML tabulky. Jednou výjimkou je hlavička, která má atribut *relative* nastaven na hodnotu *fixed*, což bylo požadavkem zadavatele. Druhou výjimkou je menu, které je při příchodu na stránku schované nad vším, co je vidět v okně prohlížeče. Atribut *position* má tedy nastaven na hodnotu *absolute*. Vzor PHP skriptu, který vytvoří prázdnou stránku o šířce 1020px, která ale obsahuje všechny prvky layoutu, naleznete v příloze C.

Obsah stránky můžeme vkládat do elementu DIV s id *content*. Zmíním zde, že místo souboru *hlavicka-bez-odkazu.php* je možné vložit soubor *hlavicka.php*. Rozdíl je v tom, že pod logem Trial se pak ještě objeví odkazy na stránky Trial. Druhým rozdílem je, že klepnutí na logo nezpůsobí zobrazení menu. Namísto toho je uživatel odkázán na úvodní stránku Frisky. Tato verze hlavičky se tedy nyní používá pouze na úvodní stránce, na

[2]CAPTCHA je test, kterým bychom měli být schopni ověřit, že uživatel je člověk. Obvykle od uživatele požadujeme, aby správně opsal deformovaný textu z obrázku.

stránce pro registraci uživatele a na stránce pro editaci profilu. Třetí verzí je soubor *hlavicka2.php*. Zde jsou vidět odkazy na Trial, ale klepnutí na logo vyvolá zobrazení menu. Tato varianta zatím není používána na žádné stránce.

Patička je také ve více variantách. Zde použitý soubor *paticka2.php* obsahuje pouze obrázek s logy Evropského sociálního fondu v ČR, Evropské unie, Ministerstva školství, mládeže a tělovýchovy, Operačního programu Vzdělávání pro konkurencechopnost. Druhá možnost - *paticka.php* - navíc obsahuje odkazy na některé rozpracované hry. Tyto odkazy jsou ale aktuálně nefunkční.

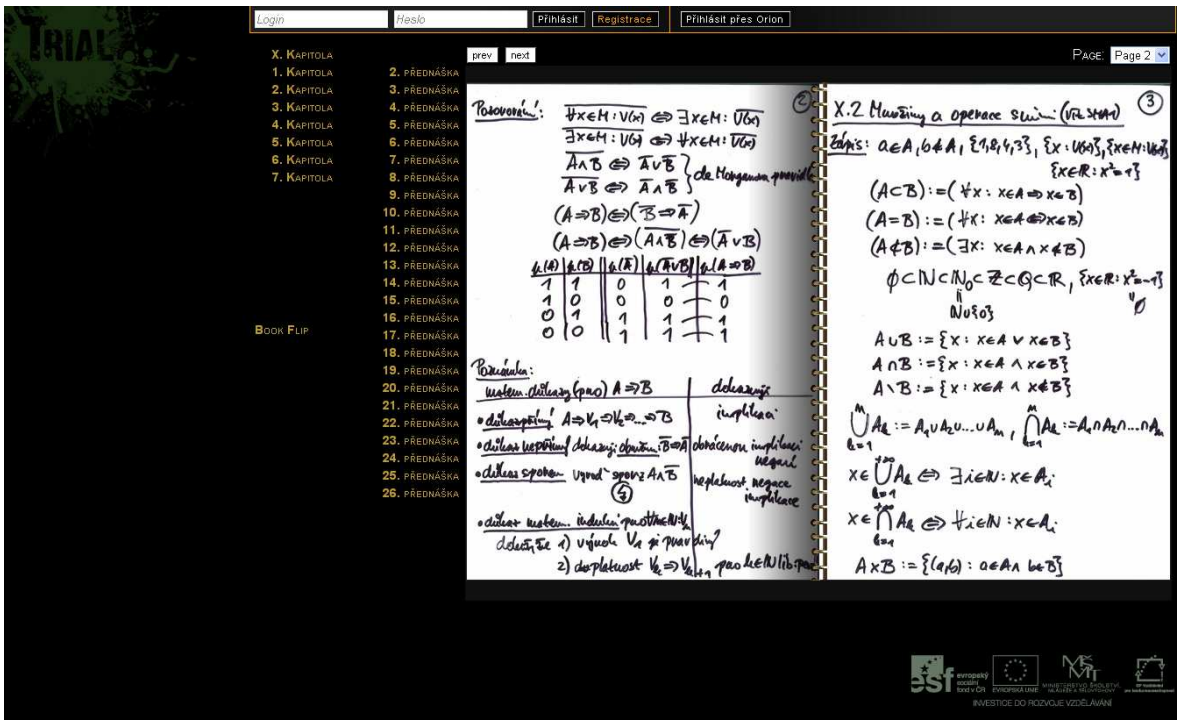
Požadavkem zadavatele bylo, že logo musí být na každé stránce vždy v levém horním rohu prohlížeče, stejně jako je to na stránkách Trial. Tedy, jak již bylo zmíněno, CSS parametr *position* hlavičky musel být nastaven na hodnotu *fixed*. Dále musel být do stránky vložen element DIV, který má stejnou šířku jako hlavička a je umístěn vlevo od obsahu stránky (jeho id je *hlavicka-space*). Ten je potřebný proto, aby „držel místo“ hlavičce. Bez něj by byla hlavička vždy schovaná pod obsahem stránky, pokud by bylo okno prohlížeče užší než obsah stránky. Poté bylo potřeba nastavit barvu pozadí obsahu stránky, aby se hlavička skutečně schovala, když bude uživatel skrolovat doprava. Nakonec jsem nastavil parametr *z-index* hlavičky a hlavní části stránky tak, aby hlavička měla *z-index* nastaven na nižší hodnotu.

Jak již bylo zmíněno, menu je při příchodu na stránku schované. Když uživatel klikne na logo stránek, vysune se menu shora a umístí se nad obsah stránky. Obsah stránky je zároveň překryt částečně průhledným DIV elementem. Ten slouží jednak estetickým účelům, ale také účelům praktickým. Tento DIV totiž způsobuje, že uživatel nemůže interagovat myší s obsahem stránky, když má zobrazené menu. To je poměrně důležité, protože nechceme, aby uživatel mohl omylem klepnout na nějaký prvek stránky ve chvíli, kdy chce používat menu. Za zmínku stojí, že jako obsah stránky je myšlen pouze element DIV s id *content* a jeho obsah. To znamená, že s ostatními prvky layoutu (s horní lištou, s patičkou atd.) je interakce možná, i když je menu zobrazené.

Vzhled stránek, kdy je menu schované, ilustruje obrázek 6.1. Stejná stránka se zobrazeným menu je vidět na obrázku 6.2.

Nevýhodou tohoto layoutu je ale jeho pevná šířka 1020 pixelů, která je pro některé stránky nedostačující. Jako řešení jsem napsal skript, který naleznete v příloze D.

Jak jsem již zmínil v analýze, bylo také potřeba vytvořit layout pro stránky, jejichž obsah je navržen tak, aby se roztahoval přes celé okno prohlížeče. Vytvořil jsem tedy velmi jednoduchý layout, kde v levé části okna prohlížeče je umístěna hlavička a ve



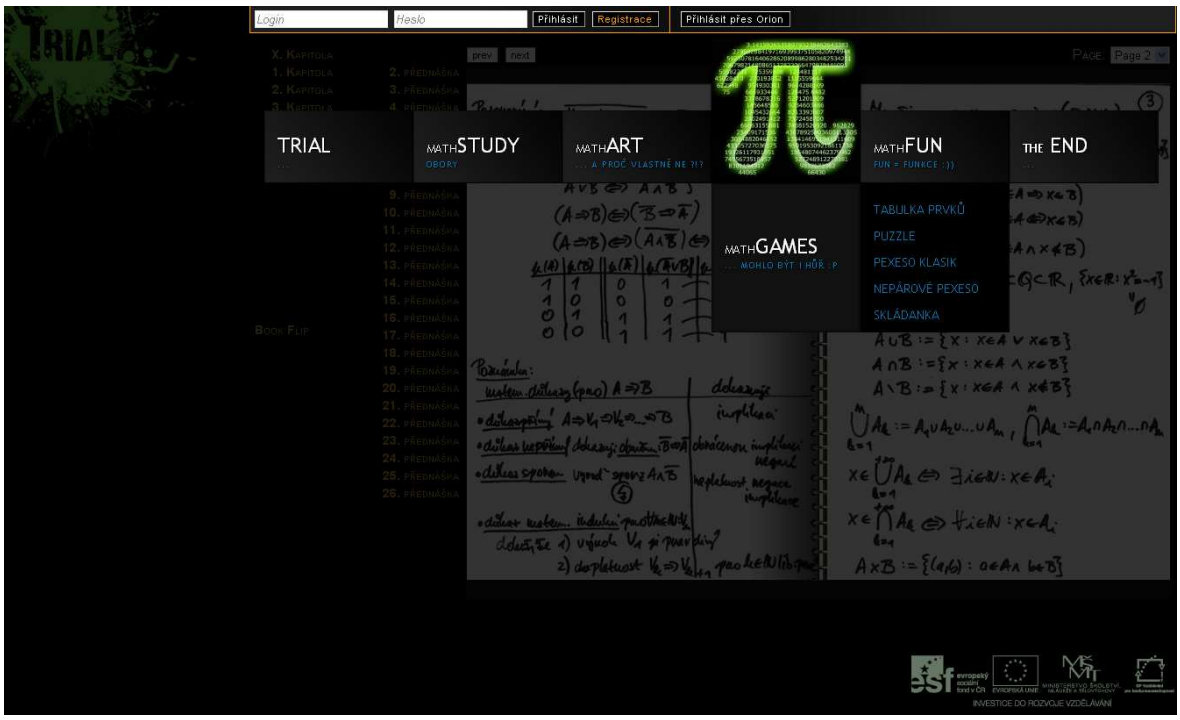
Obrázek 6.1: Stránka na Frisky, menu je schované

zbytku okna je obsah stránky. Žádné další prvky jsem do tohoto speciálního layoutu neumist'oval. Klepnutí na logo zde odkazuje na úvodní stránku Frisky. Vzor HTML těchto stránek a příslušný JavaScript naleznete v příloze E.

6.3.2 Neexistence přihlašování a registrace na Frisky

V analýze bylo uvedeno, že přihlašování na Frisky musí být možné pomocí místní registrace i pomocí systému Orion. Do horní části každé stránky jsem tedy umístil formulář pro přihlášení přes lokální registraci, dále tlačítko pro registraci a tlačítko pro přihlášení pomocí systému Orion. Po přihlášení vidí uživatel v horní liště svoje uživatelské jméno a tlačítko pro odhlášení.

Možnost lokální registrace na stránkách Frisky byla vytvořena především pro uživatele, kteří nejsou registrováni v systému Orion. Při registraci musí uživatel vyplnit celkem tři povinné údaje - uživatelské jméno, heslo a svojí e-mailovou adresu. JavaScriptový program na stránce ihned kontroluje vyplněné údaje. V případě uživatelského jména je hned po jeho vyplnění zkontrolováno, zda je volné - duplicitní uživatelská jména nejsou povolena. K docílení této funkcionality byla použita technologie AJAX. Dále heslo musí být pro kontrolu vyplněno dvakrát, program tedy hned po vyplnění



Obrázek 6.2: Stránka na Frisky, menu je zobrazené

obou políček zkontroluje, zda se hesla shodují. Heslo může být libovolné délky, jen nesmí být délka nulová. Posledním kontrolovaným údajem je e-mail. Je ověřeno, zda zadaný e-mail odpovídá formátu validní e-mailové adresy, tímto regulárním výrazem (převzato z [13]):

$$\hat{}\,[\mathrm{A–Z0-9.\_\%–}]+@[\mathrm{A–Z0-9.-}]+\backslash.[\mathrm{A–Z}]\{2,4\}\,\$$$

Uvedený regulární výraz počítá s tím, že validace není *case sensitive* (nerozhoduje velikost písmen).

Dále příslušný formulář obsahuje tři nepovinné údaje - křestní jméno, příjmení a pohlaví. Posledním prvkem formuláře je CAPTCHA. Ke generování obrázků a ověřování jejich správného opsání byla použita knihovna reCAPTCHA.

Po odeslání vyplněného formuláře na server je PHP skriptem znovu zkontrolována správnost vyplnění všech povinných údajů a také je zkontrolováno správné opsání textu z obrázku knihovnou reCAPTCHA. Je-li vše v pořádku, jsou údaje uloženy do databáze. Hesla nejsou ukládána v čisté podobě, ale jako řetězec vytvořený hashovací funkcí SHA-1. Ukládání zadaných údajů do databáze je ošetřeno proti SQL Injection použitím tzv. *prepared statement*, viz kapitola 5.3.4. Použil jsem PHP knihovnu *mysqli*

(PHP knihovna *mysql prepared statements* nepodporuje). Dále je také zamezeno HTML Injection - tj. vložení HTML značek do formuláře a jejich následné uložení do databáze. Přesněji vložit HTML značky do políček formuláře a následně je odeslat na server lze, ale PHP skript na straně serveru všechny tyto značky odstraní použitím knihovní funkce *strip_tags*.

Po zaregistrování musí uživatel svůj účet aktivovat před tím, než je mu dovoleno se přihlásit. Smysl aktivace je v ověření poskytnuté e-mailové adresy. Na zadanou e-mailovou adresu je poslán aktivační e-mail. Byla použita knihovna PHPMailer. E-mail obsahuje URL, které je potřeba navštívit pro aktivaci účtu. Aktivační URL obsahuje dva parametry: uživatelské jméno a aktivační kód. Aktivační kód je náhodně vygenerovaný řetězec, který byl při registraci uložen do databáze. Je generován tímto způsobem:

```
$code = sha1(uniqid(rand(), true));
```

Není zaručeno, že tento kód bude unikátní. Proto je jako první parametr uživatelské jméno. Po navštívení aktivační URL adresy je přečten aktivační kód z databáze, který přísluší k danému uživatelskému jménu. Tento kód je poté porovnán se zaslaným kódem. Je-li mezi nimi shoda, je do databáze uloženo, že byl daný účet aktivován. Uživatel se může poté přihlásit.

Vzhled registračního formuláře ilustruje obrázek 6.3.

Uživatel, který je přihlášen přes lokální účet na Frisky, může editovat svůj účet po kliknutí na svoje uživatelské jméno na horní liště stránek. Editovat lze pouze křestní jméno, příjmení a pohlaví. Také je možné změnit heslo. Pro změnu hesla je potřeba nejprve do příslušného formuláře zadat stávající heslo a poté dvakrát nové heslo. Vzhled stránky pro editaci profilu je vidět na obrázku 6.4.

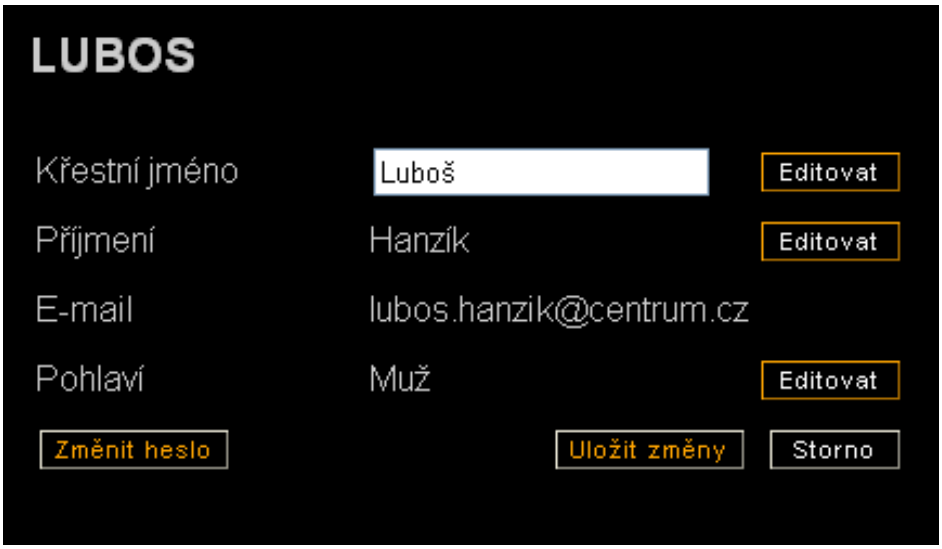
Podívejme se nyní na přihlašování pomocí systému Orion. Příslušné přihlašovací tlačítko funguje tak, že odkáže uživatele na stejnou stránku, na které se právě nachází, ale s tím rozdílem, že tentokrát se použije protokol HTTPS. Server poté uživatele automaticky přesměruje na přihlašovací stránku Orionu. Po úspěšném vyplnění přihlašovacích údajů je uživatel přesměrován na stránky Frisky. Nyní nadále komunikují server a klient protokolem HTTPS. Odhlášení je zatím možné pouze ukončením prohlížeče.

6.3.3 Stránky nejsou samostatné

Jako první krok ke splnění tohoto úkolu jsem překopíroval všechny sdílené knihovny přímo do adresáře každé stránky, která danou knihovnu potřebuje. Ve druhém kroku jsem pro každou stránku vytvořil nový PHP skript *main2.php* (Na Frisky je hlavním

Obrázek 6.3: Registrační formulář

PHP skriptem každé stránky, na který se vždy vznáší HTTP dotaz při žádosti o stránku, soubor main.php). Do souboru main2.php každé stránky jsem vložil pouze samotný obsah stránky, bez všech ostatních prvků layoutu. V tomto PHP skriptu jsem také nastavil všechny odkazy tak, aby ukazovaly pouze na zdroje, které jsou uvnitř adresáře dané stránky. Nyní lze tedy snadno překopírovat libovolnou stránku na jiný server a ihned bude možné pracovat s jejím obsahem.



Obrázek 6.4: Editace profilu uživatele

7 Analýza her

Webové stránky Frisky obsahují několik her, které jsou napsány v jazyce JavaScript. Úkolem od zadavatele bylo provést analýzu hry, která se jmenuje Puzzle. Tato analýza by měla být zaměřena na identifikaci možných způsobů podvádění v této hře. Na základě analýzy požaduje zadavatel odstranění všech v současnosti možných způsobů podvádění.

7.1 Analýza problému a návrh řešení

Hra Puzzle spočívá v tom, že uživateli je ukázán obrázek, který je následně rozdělen na 9 dílků a ty jsou pak zamíchány. Hráč musí prohazováním dílků myší složit původní obrázek. Dosažené výsledky jsou na serveru ukládány do textových souborů.

Klient provádí míchání dílků a také detekci, zda byl obrázek aktuálním tahem složen. Dále klient také měří čas hraní a počítá tahy uživatele. Když klient detekuje, že byl obrázek složen, jsou herní výsledky poslány na server, který je přijme a uloží. Vzhledem k tomu, že hra je napsána v jazyce JavaScript, uživatel si může snadno prohlédnout zdrojový kód hry a případně si jej může i upravit. Není tedy těžké si nastavit herní výsledky na libovolné hodnoty a poté nechat program tyto výsledky poslat na server, aniž by uživatel vůbec musel hru hrát.

Navrhovaným řešením těchto problémů je přesunutí všech zmíněných aktivit na server. Klient nesmí vědět, jak jsou dílky zamíchané a tudíž nesmí být ani schopen sám detekovat, zda aktuální tah vedl k úspěšnému složení obrázku.

7.2 Odstranění možnosti podvádět

Na základě předchozí analýzy jsem problém s podváděním vyřešil následujícím způsobem. Nejprve jsem přesunul zamíchání obrázku na stranu serveru. Zahájení hry tedy funguje tak, že server provede zamíchání a klient si pak z něj nahraje jednotlivé dílky, aniž by tušil, který dílek patří na kterou pozici v celkovém obrázku. Klient pouze ví, že si stahuje obrázky (dílky) 0 až 8. Server např. dostane dotaz na dílek číslo 0 a zpátky pak zašle obrázek odpovídající dílku na pozici 0, podle aktuálního zamíchání. Po dokončení této fáze může uživatel začít hrát. Nyní už není možné na straně klienta rozhodnout, zda právě provedený tah vedl k úspěšnému složení celého obrázku. Klient tedy pomocí AJAX posílá všechny tahy na server, který pak odpovídá, zda je obrázek složen nebo ne. V případě správného složení dá server klientovi toto na vědomí a navíc mu pošle i některé statistiky, tj. počet tahů a dobu řešení. Jak již bylo uvedeno v analýze, ani

počítání těchto informací není bezpečné svěřit klientovi. Poslední akcí, kterou server provede, je uložení výsledků hry.

Pro realizaci tohoto řešení bylo ale potřeba vyřešit problém s přednačítáním (preloading) obrázků. Původní implementace hry fungovala tak, že klient si pouze nahrál celý obrázek, který pak byl programově rozdělen na dílky. Nová implementace takto fungovat nemůže, protože míchání dílků provádí server. Klient si tedy stahuje každý dílek jako samostatný obrázek. Je potřeba, aby byly jednotlivé dílky staženy před tím, než začne hra. Implementace přednačítání dílků vypadá takto:

```
for(var i = 0; i < 9; i++) {
    this.tempImg[i] = new Image();
    this.tempImg[i].src = "getImage.php?imgid="+i+
      "&timestamp="+this.startTime.getTime();
    this.tempImg[i].onload = function() {
      Exploder.loadedImages = Exploder.loadedImages + 1;
      if(Exploder.loadedImages == 9) {
        Exploder.startGame();
      }
    };
}
```

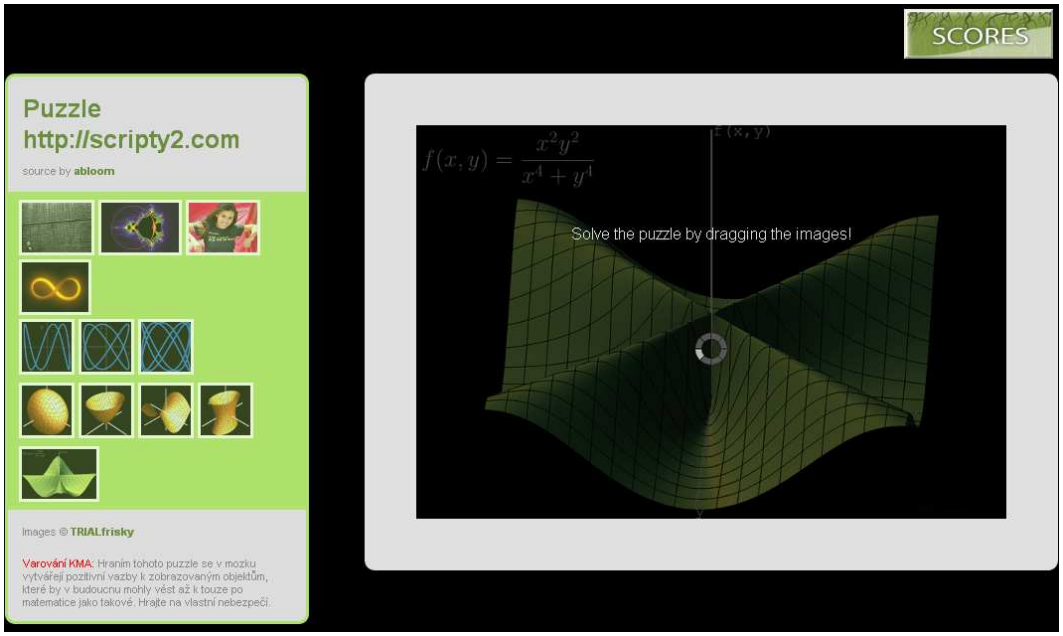
Parametrem URL obrázků je kromě identifikátoru dílku také časová značka začátku hry. Tento parametr není na serveru nijak využíván, ale je součástí URL z toho důvodu, aby byl URL obrázků při každé hře jiný. Tím je zajištěno, že klient při další hře nepoužije stejný obrázek dílku (uložený v cache), ale je nucen si jej znovu načíst ze serveru. Po stažení posledního devátého dílku se spustí hra zavoláním metody *startGame*. Na serveru se začne měřit čas hraní od chvíle, kdy byl odeslán devátý obrázek.

Je potřeba zajistit, aby během hraní byly obrázky uloženy v cache. Jinak by se přednačtené obrázky začaly v některých prohlížečích znovu načítat ve chvíli, kdy bychom je chtěli zobrazit. Proto je potřeba na serveru správně nastavit HTTP hlavičky odpovědi na žádost o obrázek:

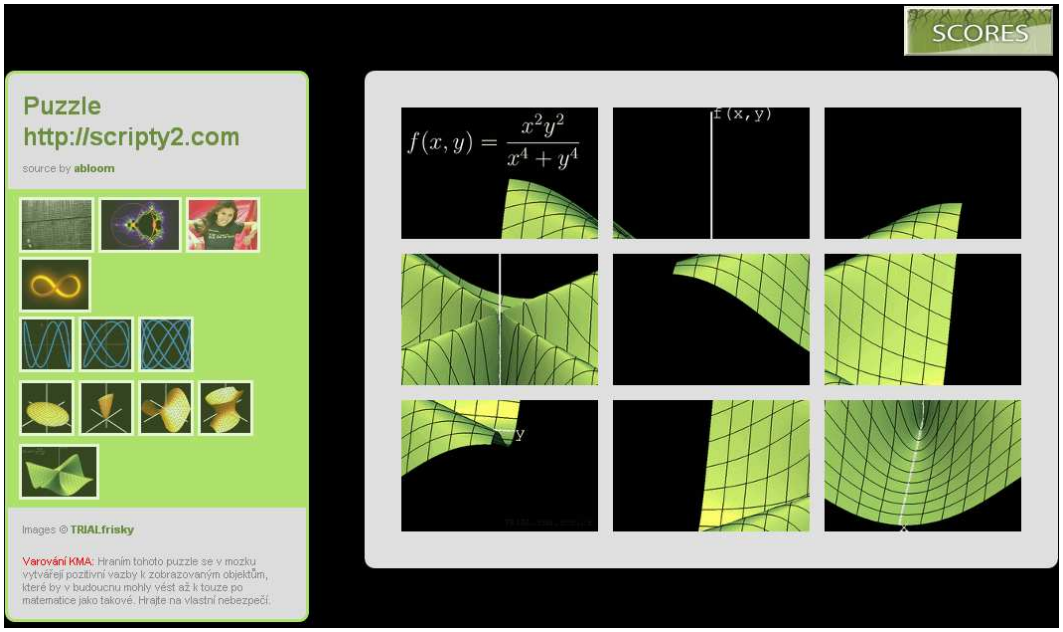
```
header('Cache-Control:');
$expires = 60*60*24*14;
header('Expires: ' . gmdate('D, d M Y H:i:s', time()+$expires) . '
  GMT');
```

Toto nastavení HTTP hlaviček prohlížeč informuje, že si má obrázky uložit do cache. Platnost obrázků vyprší až za dva týdny.

Vzhled hry Puzzle při načítání dílků ukazuje obrázek 7.1. Hra Puzzle po načtení a zamíchání obrázků je vidět na obrázku 7.2.



Obrázek 7.1: Hra Puzzle, načítání dílků



Obrázek 7.2: Hra Puzzle po načtení dílků a zamíchání

8 Rozhraní pro ukládání výsledků her

8.1 Analýza a návrh řešení

Výsledky her na Frisky se nyní ukládají do textových souborů. Pro jednodušší a efektivnější práci s nimi je vhodnější použít databázi. Zároveň je potřeba vytvořit jednotné znovupoužitelné rozhraní pro ukládání a získávání výsledků her.

Rozhraní jsem navrhl tak, že bude obsahovat pouze dvě funkce, jednu pro ukládání výsledků a druhá bude vracet nejlepší herní výsledky v podobě HTML tabulky. Dále jsem rozhodnul, že výsledky všech her budou ukládány do jedné tabulky. Výsledky jednotlivých her budou od sebe odlišeny jednoznačným identifikátorem hry. Jako herní výsledek by se měl ukládat počet dosažených bodů nebo doba řešení úkolu.

8.2 Implementace rozhraní

Vytvořil jsem rozhraní, která se skládá z následujících souborů:

- scorelib.php - obsahuje funkce pro uložení výsledků a získání nejlepších výsledků v HTML podobě.
- scorelib.css - obsahuje nastavení vzhledu tabulky, ve které se zobrazují nejlepší výsledky.
- scorelib.js - obsahuje funkci, která je volána, když uživatel klikne na některou ze záložek v tabulce výsledků.
- scorebox.png - pozadí tabulky výsledků.

Pro ukládání výsledků byla zvolena databáze MySQL. Podle předchozí analýzy jsou výsledky her ukládány do jedné tabulky. Pojmenoval jsem ji *gameResults*. Jako výsledek hry je možné ukládat bodové skóre nebo dobu, za kterou hráč vyřešil danou úlohu, případně obojí. Podle toho je potom také možné při zobrazení nejlepších herních výsledků nechat zobrazit dosažené skóre, dobu řešení nebo obojí. Tabulka s nejlepšími výsledky může být řazena také podle libovolného z těchto ukazatelů, a to sestupně nebo vzestupně.

Soubor *scorelib.php* obsahuje celkem dvě funkce, které bude uživatel knihovny volat:

- saveResults($gameID, $playerName, $score, $gameDuration, $otherInfo) - slouží k ukládání výsledků her. Jednotlivé parametry mají následující význam:

  - *$gameID* - Celočíselný identifikátor hry, jejíž výsledky chceme ukládat.
  - *$playerName* - Uživatelské jméno uživatele, jehož herní výsledek ukládáme.
  - *$score* - Dosažené skóre. Musí se jednat o celé číslo.
  - *$gameDuration* - Doba řešení úlohy. Musí být udána také celým číslem jako počet milisekund.
  - *$otherInfo* - Sem je možné uložit libovolné další informace jako řetězec délky maximálně 300 znaků.

- getBestResults($game, $timePeriod, $order, $columns) - slouží k získání nejlepších herních výsledků, které jsou ve formě HTML odeslány klientovi. Parametry mají tento význam:
  - *$game* - Číselný identifikátor hry.
  - *$timePeriod* - Vybírá, za jaké časové období chceme zobrazit nejlepší výsledky. Může nabývat tyto hodnoty:
    * 1 - Nejlepší výsledky za tento den.
    * 2 - Nejlepší výsledky za tento týden.
    * 3 - Nejlepší výsledky za tento měsíc.
    * 4 - Nejlepší výsledky všech dob.
    * 5 - Nejlepší výsledky všech dob, ale každý uživatel zde má nejvýše jeden (svůj nejlepší) výsledek.
    * 6 - Nejlepší výsledky za posledních 30 dní.
  - *$order* - Udává, jak mají být výsledky řazeny. Lze řadit podle dosaženého času nebo získaného skóre. Knihovna definuje následující konstanty:
    * $ORDER_BY_TIME_DESC - seřadí výsledky sestupně podle dosaženého času.
    * $ORDER_BY_TIME_ASC - seřadí výsledky vzestupně podle dosaženého času.
    * $ORDER_BY_SCORE_DESC - seřadí výsledky sestupně podle dosaženého skóre.
    * $ORDER_BY_SCORE_ASC - seřadí výsledky vzestupně podle dosaženého skóre.
  - *$columns* - Říká, které sloupce by měly být zobrazeny v tabulce výsledků jako ukazatele dosaženého herního výsledku. Je možné zobrazit dosažený čas nebo dosažené skóre nebo obojí. Knihovna definuje příslušné konstanty:
    * $SHOW_TIME - zobrazí se dosažený čas.
    * $SHOW_SCORE - zobrazí se dosažené skóre.
    * $SHOW_BOTH - zobrazí se dosažený čas i dosažené skóre.

Ukázku použití této knihovny naleznete v příloze F.

9 Knihovna pro komunikaci uživatelů

Úkolem od zadavatele bylo vytvořit prostředky pro komunikaci uživatelů na webových stránkách. Prvním způsobem komunikace, který měl být implementován, bylo kreslení na HTML5 element canvas. Kreslit by se mělo myší. Vše, co některý z návštěvníků stránky nakreslí, by se mělo zobrazit u všech ostatních návštěvníků. Zadavatel si také přál, aby bylo možné nakreslené obrázky ukládat do galerie. Dalším způsobem komunikace by měla být textová komunikace (chat). Knihovna by také měla umožnit zobrazit seznam uživatelů, kteří jsou přítomni na stránce.

9.1 Analýza distribuce aktualizací

Hlavním problémem, který zde musíme rozebrat, je způsob distribuce napsaných zpráv nebo nakreslených čar (dále označujme jednoduše jako aktualizace) od jejich autora k ostatním uživatelům. Abychom docíli této funkcionality, bude nejprve potřeba zajistit ukládání všech aktualizací na serveru. Klient bude muset všechny aktualizace posílat na server prostřednictvím technologie AJAX. Na serveru bude nejlepší aktualizace ukládat do databáze.

Dále je potřeba rozhodnout, jak budou klienti získávat aktualizace, které na serveru někdo jiný uložil. Jedním možným řešením by mohlo být posílání dotazů na server v pravidelných intervalech. Těmi bychom se serveru ptali, zda se objevila v databázi aktualizace, kterou ještě nemáme. Taková implementace by ale nebyla příliš optimální, nebot', pokud chceme aktualizace dostávat s malým zpožděním, museli bychom server zatěžovat častými dotazy. Z toho důvodu jsem se rozhodl zvolit lepší řešení - použít programátorský model, kterému se říká *Comet*. Konkrétně jsem zvolil jeho variantu *long polling* s použitím technologie AJAX.

Metoda *long polling* funguje tak, že klient se dotáže prostřednictvím AJAX dotazu serveru, zda je v databázi něco nového. Pokud ano, skript pošle klientovi příslušná data a skončí. Pokud ne, skript na serveru hned neodpoví, ale ptá se v pravidelných intervalech databáze, zda je něco nového. Klientovi je poslána odpověď až ve chvili, kdy se objeví nějaká aktualizace. Jakmile klient dostane odpověď, zpracuje si získaná data a pošle na server nový dotaz. U této metody bude kritické zajistit, aby příslušný PHP skript vždy ukončil svůj běh poté, co uživatel opustí stránku. Tento skript se tedy bude muset vždy po určité maximální době běhu ukončit a klient pak bude muset obnovit spojení. Bylo by sice možné v javascriptu reagovat na události *unload* a *beforeunload*, ale takové řešení by nebylo spolehlivé.

Metodu *long polling* bude možné použít pro kreslení, chat, galerii i pro získávání aktuálního seznamu přihlášených uživatelů přítomných na stránce.

Nyní je potřeba rozmyslet, jak budeme udržovat seznam přihlášených uživatelů a jak budeme detekovat, že uživatel opustil stránku. Řešení druhého zmíněného problému jsem už uvedl výše. Od každého uživatele budou muset přicházet pravidelně nové dotazy. Když nedostaneme dotaz do určité doby od posledního dotazu, budeme uživatele považovat za nepřítomného. Seznam přítomných uživatelů musíme mít uložený tak, aby s ním mohly pracovat všechny procesy. Bude tedy potřeba použít např. databázi nebo sdílenou paměť.

Další problém, který rozebereme, je, kdy by měl klient poslat aktualizaci na server. V případě chatu a galerie je toto jasné - aktualizaci odešleme, když uživatel klepne na tlačítko „Odeslat zprávu“ nebo „Uložit obrázek“. V případě kreslení se ale nabízí více možností. Jednou možností by bylo poslat na server aktualizaci vždy, kdy i u místního klienta voláme překreslení canvasu. To znamená vždy, kdy nám přibyde nový bod kreslené čáry. Tento přístup by byl z hlediska efektu pro uživatele nejhezčí. Je ale jasné, že by se potom na server posílalo extrémně vysoké množství požadavků. Další možností je posílat data na server vždy, kdy uživatel dokreslí danou čáru. To znamená ve chvíli, kdy pustí levé tlačítko myši. Toto bych považoval za rozumné řešení. Bylo by ale možné jej vylepšit posíláním čar po částech v případě, kdy uživatel kreslí dlouhou čáru. Např. by se aktualizace posílala po nakreslení každých dvaceti bodů čáry. U tohoto řešení by ale bylo implementačně náročné udržet konzistence obrázků tak, aby čára započatá později byla i se všemi svými úseky kreslena přes čáru započatou dříve (a přes všechny její později dodané úseky).

9.2 Implementace metody long polling

Podle předchozí analýzy jsem pro distribuci aktualizací implementoval metodu long polling. Princip této metody byl již popsán v analýze - klient musí poslat technologií AJAX dotaz na server, odpověď dostane až když server nalezne v databázi novou aktualizaci. Pro každého uživatele tedy běží na serveru jeden PHP skript (s tím, že vždy po nějaké době skončí a po příchodu nového požadavku se spustí znovu), který se pravidelně dotazuje databáze. Pokud je na stránce přítomno mnoho uživatelů, mohou tyto pravidelné databázové dotazy způsobit znatelné zatížení serveru. Uživatel knihovny tedy bude muset nastavit jejich periodu. Při delší periodě se sníží zatížení serveru, ale zase se prodlouží doba, za kterou se aktualizace rozešle uživatelům.

V analýze bylo také zmíněno, že musí být ošetřeno, aby PHP skript nesměl za žádných okolností běžet nekonečně dlouho. Nebezpečí vzniká především tím, že skript

pravidelně volá funkci *set_time_limit*, čímž si zajišťuje potenciálně nekonečnou dobu běhu, server jej nemůže automaticky ukončit. Provedl jsem tedy tato opatření:

- Timeout. Skript se ukončí vždy nejpozději po 45 sekundách a klient musí spojení obnovit.
- Použití událostí *beforeunload* a *unload*. Jejich použití není příliš spolehlivé, neboť zatímco například událost *unload* nastane v některých prohlížečích při odchodu ze stránky i při zavření příslušného okna/tabu, v jiný nastane pouze při odchodu ze stránky. Událost *beforeunload* některé prohlížeče neznají vůbec. Pokud je ale některá z těchto událostí vyvolána, pošle se na server zpráva, že skript, který pro nás zjišťuje přítomnost nové aktualizace, se má ihned ukončit.
- PHP skript na serveru také umí detekovat, že pro stejného uživatele běží další skript, který by mu také měl dodávat aktualizace. Tato situace například nastane, když uživatel provede opětovné načtení (reload) stránky. Pokud skript tedy tuto situaci detekuje a zjistí, že je starší, než druhý běžící skript, sám se ukončí. Této funkcionality jsem dosáhl použitím sezení (session), kam si skript vždy uloží čas, kdy začal běžet. Když se skript ukončuje, vždy si tento záznam smaže. Pokud tedy při svém běhu skript zjistí, že takové záznamy jsou v sezení dva (nebo více) a že některý z těchto záznamů obsahuje vyšší časovou značku, než je značka uložená daným skriptem, pak se skript ukončí, neboť ví, že existuje jiný novější skript.

Dále bylo potřeba vyřešit problém s použitím sezení. Při volání PHP funkce *session_start* se skript zároveň snaží získat binární zámek sezení, který je standardně uvolněn po ukončení skriptu. V našem případě chceme, aby skript mohl běžet dlouhou dobu a zároveň mohl používat sezení. Pokud by ale náš skript uzamknul sezení, nemohl by uživatel posílat další požadavky na jiné PHP skripty, které také používají sezení. Např. by tedy uživatel nemohl přejít na jinou stránku (v dané doméně) do doby, než by se skript sledující aktualizace ukončil. Naštěstí nabízí PHP funkci *session_write_close*, která ukončí práci se sezením a vzdá se zámku. Tuto funkci tedy volám vždy než se skript uspí. Po probuzení musí náš skript znovu zavolat funkci *session_start*, aby mohl opět používat sezení. Toto ovšem přináší další problém, neboť PHP při volání této funkce vždy automaticky zapíše do HTTP hlavičky odpovědi cookie PHPSESSID. Kolikrát během svého běhu skript zavolá *session_start*, tolikrát se v HTTP hlavičce (v parametru Set-Cookie) tato cookie objeví (vždy se stejnou hodnotou). Kromě toho, že pak posíláme zbytečně dlouhou odpověď, může toto také způsobit problémy v některých prohlížečích. Řešením je přepsat parametr Set-Cookie takto:

```
header('Set-Cookie: '.SID.'; path=/', true);
```

9.3 Kreslení

9.3.1 Implementace kreslení v elementu canvas

V této podkapitole popíši, jak jsem implementoval kreslení v elementu canvas, aniž bych se zatím zabýval distribucí nakreslených čar k ostatním uživatelům.

Vytvoření elementu canvas provádím následovně:

```
canvasWidth = "600px";
canvasHeight = "400px";

var canvasDiv = document.getElementById('canvasDiv');
canvas = document.createElement('canvas');
canvas.setAttribute('width', canvasWidth);
canvas.setAttribute('height', canvasHeight);
canvas.setAttribute('id', 'canvas');
canvasDiv.appendChild(canvas);

context = canvas.getContext("2d");
oldimage = context.getImageData(0, 0, canvasWidthInt,
    canvasHeightInt);
```

Zde je důležitá předposlední řádka. Zavoláním metody *getContext* dostaneme ukazatel na objekt reprezentující 2D kontext canvasu. Ten dále použijeme k tomu, abychom mohli na canvas kreslit. Poslední řádka je také důležitá, zde si totiž ukládáme binární reprezentaci obrázku na canvasu. Tu budeme dále využívat při překreslování obrázku.

Nyní se podívejme, jak vlastně funguje kreslení na element canvas, aniž bychom se zatím zabývali distribucí nakreslených čar na ostatní počítače.

Na element canvas kreslíme myší. Pokud je stisknuto tlačítko myši, kreslí se čára podle pohybu kurzoru myši. Nad elementem canvas tedy potřebujeme sledovat následující události:

- *mousedown* - při této události si uložíme, že právě teď kreslíme. Poté si uložíme aktuální souřadnice kurzoru myši jako první bod kreslené čáry a provedeme překreslení.
- *mousemove* - pokud máme uloženo, že právě teď kreslíme (je stisknuté tlačítko myši), potom si uložíme aktuální pozici kurzoru myši jako další bod kreslené čáry a překreslíme canvas.

- *mouseup* - uložíme si, že kreslení aktuální čáry bylo dokončeno. Následující události mousemove jsou tedy ignorovány do doby, než opět uživatel stiskne levé tlačítko myši nad elementem canvas. Tato událost je sledována nad elementem *body*. Je tomu tak pro případ, kdy uživatel při kreslení sjede s myší z elementu canvas a pak pustí levé tlačítko myši.

Podívejme se zjednodušeně na to, jak funguje funkce zajišťující překreslení canvasu. Tato funkce je v programu pojmenována jako *redraw*. Pokud by se při kreslení vždy pouze vykreslila další část čáry, prohlížeč by neprovedl antialiasing obrázku. Čáry by potom byly nevzhledně kostrbaté. Nejprve tedy překreslíme canvas jeho binární reprezentací, kterou jsme si uložili při posledním překreslení. Překreslení obrázku provedeme takto:

```
context.putImageData(oldimage, 0, 0);
```

Následně budeme kreslit na canvas podle toho, co uživatel právě provedl. Kreslení začneme zavoláním funkce *beginPath*:

```
context.beginPath();
```

Tato funkce vyčistí seznam všech primitiv, která jsme doposud nakreslili. Další postup je závislý na tom, zda právě začínáme kreslit novou čáru, nebo zda pouze kreslíme spojnici s dalším bodem už započaté čáry. Pokud pokračujeme v kreslení započaté čáry, musíme použít funkci *moveTo*, abychom naše neviditelné pero přesunuli na místo, které budeme spojovat s novým bodem čáry. Pokud začínáme s novou čarou, přesuneme se na její začátek, tj. na aktuální pozici myši.

Uvažujme nyní tedy, že začínáme kreslit novou čáru. V tuto chvíli musíme vykreslit tečku v uložené aktuální pozici myši:

```
context.arc(clickX[i], clickY[i], clickSize[i]/2.0, 0, Math.PI*2,
    true);
context.closePath();
context.fillStyle = clickColor[i];
context.fill();
```

Všechna pole, která jsou vidět v této ukázce kódu, obsahují informace o právě kreslené čáře. Z polí *clickX* a *clickY* si přečteme aktuální souřadnice myši. V poli *clickSize* máme uloženou aktuální šířku čáry a v poli *clickColor* aktuální zvolenou barvu. Kružnice se vykreslí funkcí *arc*, její vyplnění je pak provedeno funkcí *fill*.

Pokud chceme vykreslit spojnici s dalším bodem čáry, použijeme místo funkce *arc* funkci *lineTo* a místo funkce *fill* zavoláme *stroke*:

```
context.lineTo(clickX[i], clickY[i]);
context.closePath();
context.strokeStyle = clickColor[i];
context.lineWidth = clickSize[i];
context.stroke();
```

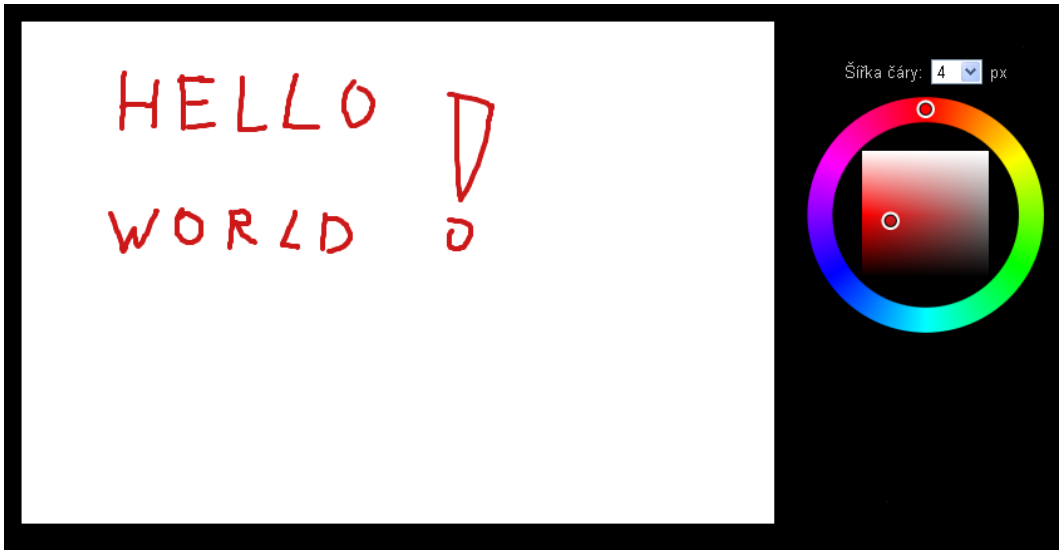
Následně si po vykreslení nového obrázku uložíme jeho binární reprezentaci, kterou použijeme při dalším překreslení obrázku. To provedeme následovně:

```
oldimage = context.getImageData(0, 0, canvasWidthInt,
    canvasHeightInt);
```

Z výše uvedených příkladů je vidět, že uživateli je dovoleno zvolit si barvu a šířku kreslené čáry. Knihovna je implementována tak, že její uživatel si může nadefinovat rozbalovací seznam (elementem *select*), ve kterém si bude moci uživatel kreslení volit šířku čáry v pixelech. Pro výběr barvy lze použít např. některý z volně dostupných modulů do knihovny jQuery. Ve své ukázce použití knihovny, která je nasazena na stránkách Frisky, jsem použil plug-in *farbtastic*.

Výsledné kreslení na HTML5 element canvas (spolu s použitým *farbtastic* modulem) ukazuje obrázek 9.1.

Tato implementace kreslení byla částečně založena na návodu v [14].



Obrázek 9.1: Kreslení

9.3.2 Získávání aktualizací od ostatních

Získávání aktualizací je založeno na již popsané implementaci metody *long polling*. Podívejme se, jaké informace se vyměňují mezi klientem a serverem. Server vždy posílá s aktualizacemi také identifikátor nejnovější posílané aktualizace. Klient si tento identifikátor ukládá do příslušné proměnné. Identifikátor odpovídá sloupci *id* v příslušné tabulce databáze, který je nastaven takto: *BIGINT UNSIGNED NOT NULL AUTO_INCREMENT*. Jedná se tedy o automaticky inkrementovaný číselný identifikátor řádku tabulky. Číslo posledně obdrženého identifikátoru aktualizace klient vždy posílá jako parametr svého požadavku o aktualizaci. Při prvním požadavku posílá hodnotu nula. Pak skript na serveru začne načítat z databáze všechny poslední aktualizace obrázku, dokud součet délek čar, které tyto aktualizace obsahují, nepřekročí určitou maximální hodnotu, nebo dokud nenačte všechno, co je v databázi. Všechny načtené aktualizace jsou pak poslány klientovi a zde vykresleny na canvas. Tím by měl tedy uživatel po příchodu na stránku vidět, co ostatní uživatelé doposud nakreslili.

Po tomto úvodním načtení obrázku pošle klient nový dotaz na server. Nyní už budou klientovi zasílány pouze ty aktualizace, které mají v databázi vyšší identifikátor, než jaký měla poslední klientem obdržená aktualizace.

Dále je potřeba zdůraznit, že po uložení aktualizace obrázku je tato detekována všemi skripty, které posílají uživatelům aktualizace. Vyjímkou by ale měl být skript příslušející samotnému autorovi aktualizace. Jak bude dále podrobněji rozebráno, je to zajištěno tím, že při příjmu aktualizace si server pomocí session zapamatuje, že danou aktualizaci má její autor vykreslenou a není potřeba mu ji posílat znovu. Může ale nastat jedna vyjímka. Jedná se o situaci, kdy server přijal aktualizaci od jiného uživatele těsně předtím, než uložil naši aktualizaci. Problém je v tom, že pro udržení konzistence obrázků by se cizí aktualizace měla vykreslit pod tu naši (pokud se čáry překrývají). Tuto speciální situaci server vyřeší tak, že nám pošle cizí i naši aktualizaci. Klient je pak obě vykreslí v tomto pořadí a konzistence obrázků je tím zachována.

9.3.3 Uložení dat na serveru

Podle předchozí analýzy jsem se rozhodl posílat aktualizace na server vždy po dokreslení dané čáry, neboť toto řešení považuji za nejrozumnější z hlediska funkčnosti i implementační složitosti. K odeslání dat na server používám technologii AJAX a nad ní AjaxManager, což je přídavný modul do knihovny jQuery. Tento modul používám, abych zajistil, že jednotlivé požadavky budou na serveru zpracovány ve správném pořadí. AjaxManager totiž umí zajistit odeslání nového požadavku až po získání odpovědi na ten předchozí. Kód, který vloží AJAX požadavek do fronty, ze které bude vybrán a odeslán až na něj přijde řada, vypadá následovně:

```
drawingAjaxManager.add({
  url: url,
  type: 'POST',
  beforeSend: function(jqXHR, settings) {
    sendingDrawingUpdate = true;
    settings.data += "&lastid="+encodeURIComponent(lastid);
  },
  data:{"msg":msg},
  success: function(transport) {
    var inc = parseInt(transport);
    if(inc > 0) {
      lastid = lastid + inc;
    }
  },
  complete: function() {
    sendingDrawingUpdate = false;
    processQueuedCanvasUpdates();
  }
});
```

Data jsou posílána metodou POST HTTP protokolu. Velikost dat může být totiž relativně velká. Záleží na délce nakreslené čáry. Na server je posílán také identifikátor (dále zkráceně id) poslední zobrazené aktualizace. Pokud aktualizace, kterou se nyní snažíme uložit, bude mít v databázi id o jedno vyšší, server nás o tom informuje v odpovědi na tento AJAX požadavek. Klient si pak podle toho může inkrementovat svojí proměnnou, ve které uchovává id poslední zobrazenené aktualizace. Pokud byla těsně před naší aktualizací uložena aktualizace cizí, id inkrementovat nemůžeme, ale pravděpodobně by nám měla brzo přijít zpráva, která bude obsahovat obě aktualizace. Ve skutečnosti, pokud právě odesíláme aktualizaci na server, zpracovnání příchozí aktualizace je odloženo a provede se po dokončení odesílání naší aktualizace (viz callback funkce jako hodnota parametru *complete* v AJAX požadavku). Je tomu tak, aby při dalších požadavcích na server byl parametr id nastaven na správnou hodnotu podle toho, jakou poslední aktualizaci máme skutečně zobrazenou.

Naprogramoval jsem celkem tři formáty aktualizací, které lze na server posílat.

- *Obecný formát* - Tento typ formátu zpráv slouží k posílání více čar najednou. Ty jsou reprezentovány jako série bodů. Každý bod je reprezentován jeho souřadnicemi. Dále je ke každému bodu uloženo, zda má být spojen s přechozím bodem, a případně jak tlustou čarou a jaká má být barva této čáry. Tento formát zpráv je nyní použit na stránkách Frisky pro automatické vykreslení souřadných os po stisknutí příslušného tlačítka na stránce.

- *Speciální formát pro kreslení* - Slouží k uložení informací o jedné jednobarevné čáře jednotné tloušťky. To je ideální pro potřeby kreslení myší, kdy se aktualizace posílají vždy po dokreslení jedné čáry. Formát tedy obsahuje barvu a tloušťku čáry a souřadnice všech bodů, jejichž spojením se čára vykreslí.

- *Smazání obrázku* - Tento poslední typ zpráv slouží k uložení informace o tom, že se má smazat obsah canvasu. Mazání by šlo provést i jednoduše použitím jednoho z předchozích typů formátů, kdy bychom zkrátka uložili bílou čáru, která překreslí vše, co je nyní na canvasu. Nakonec to i při příjmu této zprávy klient provede. Speciální formát jsem ale použil k tomu, aby server mohl při posílání aktualizací snadno detekovat, že daná aktualizace maže vše na canvasu. Není tedy potřeba, aby posílal cokoliv, co bylo uloženo před ní.

Na serveru jsou data ukládána bez ohledu na použitý formát do stejné tabulky databáze. Data poslaná v jedné zprávě (typicky obsahující informace o jedné nakreslené čáře) jsou uložena do jedné řádky tabulky. Data poslaná v obecném formátu jsou uložena jako MySQL typ *MEDIUMTEXT* do jednoho sloupce tabulky. Zvlášť je uložena pouze délka čár a typ zprávy. V případě speciálního formátu pro kreslení ukládáme do jednoho sloupce souřadnice všech bodů. Barvu a tloušťku čáry ale ukládám zvlášť do jiných sloupců tabulky. U zprávy informující o smazání canvasu ukládám do databáze pouze příslušný typ zprávy.

9.4 Galerie

Galerie také používá metodu *long polling*. Ta je použita k počátečnímu naplnění galerie a pak k získávání nově uložených obrázků. Po příchodu na stránku se do galerie stáhne určitý počet nejnovějších obrázků (záleží na nastavení knihovny). Na žádost uživatele je možné do galerie stáhnout i starší obrázky. Ke stahování starších obrázků se už pochopitelně *long polling* nepoužívá. Klient od serveru vždy nejprve získává pouze informace o obrázcích, tj. identifikátor obrázku, popis, přezdívku autora a čas uložení. Získané identifikátory jsou pak použity klientem k sestavení URL jednotlivých obrázků. Tyto URL jsou pak použity ke stažení obrázků. JavaScript obrázky stahuje popořadě. Při počátečním naplnění a při stahování starších obrázků se obrázky stahují od nejnovějšího po nejstarší. Při stahování nově uložených obrázků je tomu naopak. Po stažení prvního obrázku je tento vložen do galerie a začne se stahovat další obrázek atd. K načítání obrázků je tedy použita technika přednačítání obrázků (image preloading). Zdrojový kód pro přednačtení obrázku vypadá takto:

```
img = new Image();
img.onload = function() {
  putImageIntoGallery();
```

```
};
img.src = "getImage.php?id="+currid;
```

Tento kód začne přednačítat jeden obrázek. Po dokončení načítání je tento vložen do galerie funkcí *putImageIntoGallery*. Ta jako poslední operaci zavolá znovu funkci obsahující výše uvedený kód, čímž se začne stahovat další obrázek v pořadí.

Uživatel knihovny si musí nastavit rozměry galerie v počtu obrázků a také rozměry samotných obrázků v galerii v počtu pixelů. V současnosti je knihovna implementována tak, že jsou obrázky stahovány v plné velikosti, jejich zmenšení na požadovanou velikost provádí webový prohlížeč na základě atributů *width* a *height* elementu *img*. Kolik obrázků se stáhne do galerie po příchodu na stránku nastavuje také uživatel knihovny. Pod galerií je umístěn nápis „Více“. Když na něj uživatel klepne, stáhnou se do galerie další starší obrázky. Stáhne se jich tolik, aby celkový počet obrázků v galerii byl násobkem velikosti galerie (aktuální počet obrázků nemusí být vždy násobkem velikosti galerie, neboť do galerie přibývají nově uložené obrázky hned po jejich uložení). Na stránkách Frisky se po klepnutí na libovolný obrázek v galerii nejprve schová kreslící plátno (canvas) a další ovládací prvky, a na jejich místě se daný obrázek zobrazí v plné velikosti. Pokud chceme obrázek zase schovat, stačí na něj klepnout myší (je jedno, jestli na jeho zvětšenou verzi, nebo na jeho menší verzi v galerii). Ošetření toho, co se má stát po klepnutí na obrázek v galerii, není ale součástí knihovny a měl by si toto její uživatel naprogramovat podle svého uvážení.

Informace o obrázku, tj. uživatelské jméno autora (pokud obrázek uložil nepřihlášený uživatel, je nastaveno jako *Anonymous*), datum uložení a popisek, zobrazuji jako tooltip po najetí myší na daný obrázek v galerii. K vytvoření těchto tooltipů byla použita knihovna qTip 2.0, viz podkapitola 2.5.6.

Ukládání obrázků je naprogramováno tak, že klient získá reprezentaci obrázku na elementu canvas použitím funkce *toDataUrl*:

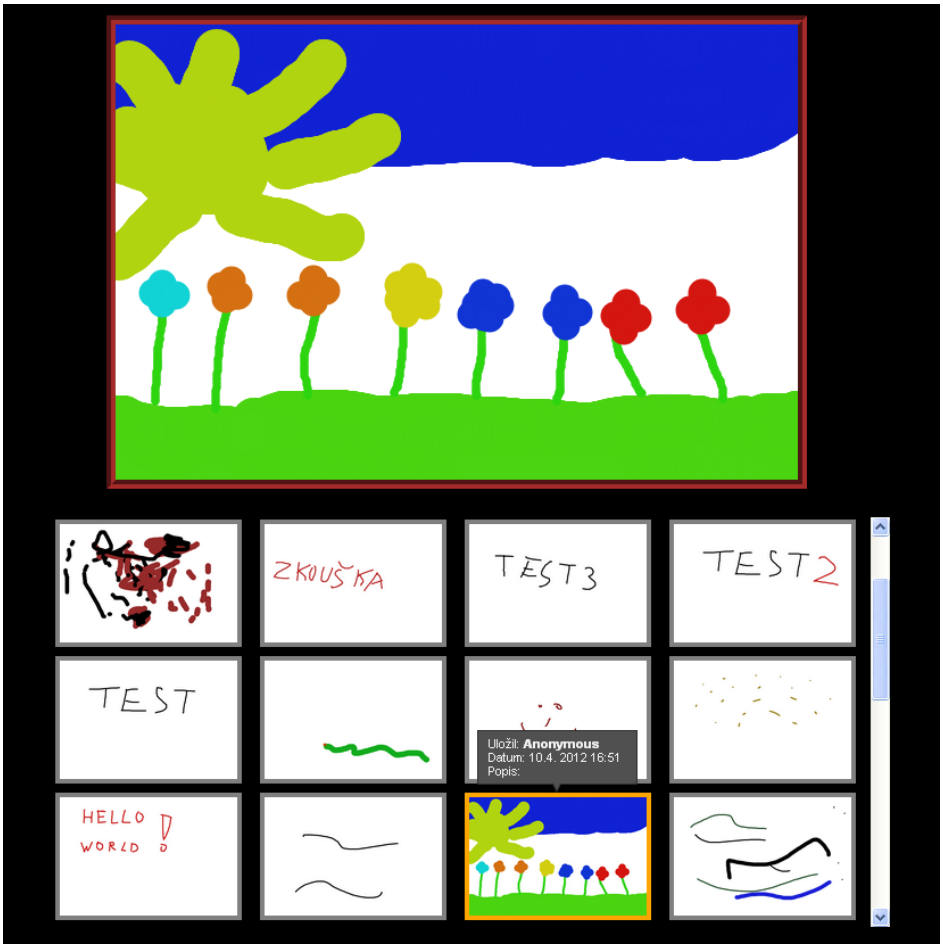
```
document.getElementById("canvas").toDataURL();
```

Data jsou získána ve formátu „data URL“, který je popsán v RFC 2397. Tato data jsou následně na server zaslána AJAX dotazem. Na serveru jsou data převedena z textové podoby do binární odstraněním nepotřebného začátku obdrženého řetězce, který obsahuje deklaraci typu dat, a následným použitím PHP funkce *base64_decode*:

```
$image = base64_decode( str_replace('', '',
    $dataUrl));
```

Získaný PNG obrázek je pak uložen do databáze.

Vzhled galerie na stránkách Frisky si můžete prohlédnout na obrázku 9.2.



Obrázek 9.2: Galerie

9.5 Chat

Chat je opět založen na metodě *long polling*. Po příchodu na stránku je uživateli zobrazena historie posledních zpráv. Maximum zpráv, které jsou uživateli na počátku poslány, definuje uživatel knihovny. Stejně tak definuje, jak daleko do minulosti může tato historie sahat. Získávání zpráv od ostatních uživatelů funguje podobně jako získávání aktualizací kreslení. Jediným rozdílem je to, že se nezajišťuje konzistentnost pořadí zobrazených zpráv. Je tomu tak, protože jsem chtěl, aby zpoždění mezi odesláním zprávy a jejím zobrazením v chatu autora zprávy bylo nulové. To znamená, že klient nečeká při odeslání zprávy na potrvzení od serveru, že těsně před přijetím naší zprávy nepřijal zprávu od někoho jiného, která by se tedy pro zachování konzistence měla zobrazit v chatu nad naší zprávou. Vložit takovou zprávu dodatečně nad tu naši by také nebylo vhodné, neboť pak by ji mohl uživatel přehlédnout. Toto rozhodnutí

jsem učinil, neboť si myslím, že tato nekonzistentnost nezpůsobuje žádné problémy. Zprávy od ostatních uživatelů budou zobrazeny vždy v takovém pořadí, v jakém je server uložil.

V chatu je vždy před zprávou zobrazené uživatelské jméno uživatele, který je jejím autorem, a čas odeslání zprávy. Pokud není autor zprávy přihlášen, je místo uživatelského jména vypsáno slovo *Anonymous*.

Jméno autora a čas u zprávy mají zelenou barvu, pokud je uživatel autorem dané zprávy. V opačném případě mají červenou barvu. Pro přihlášené uživatele toto platí i v zobrazené historii chatu po opětovném načtení stránky. V tomto ohledu je knihovna přizpůsobena speciálně pro fungování na stránkách Frisky, kde je povoleno být přihlášen přes systém Orion nebo přes místní registraci. Když uživatel napíše zprávu, je do databáze také uloženo, jakým způsobem byl její autor přihlášen. Uživatel je považován za přihlášeného, pokud nalezneme jeho uživatelské jméno buď v proměnné $_SESSION["login"] (místní registrace) nebo v proměnné $_SERVER['WEBAUTH_USER'] (Orion login). JavaScript také potřebuje vědět o tom, jaké je uživatelské jméno místního uživatele. Rozhodl jsem, že v případě, kdy je uživatel přihlášen přes místní registraci, bude uživatelské jméno vypsáno v HTML elementu s id *usernameLocal*. Pokud je uživatel přihlášen přes systém Orion, mělo by být uvnitř elementu s id *usernameOrion*.

Chat je ošetřen proti SQL a HTML injection stejným způsobem jako např. formuláře pro registraci na stránkách Frisky, tj. použitím *prepared statements* a PHP funkce *strip_tags*.

9.6 Zobrazení přihlášených uživatelů

Poslední částí knihovny je zobrazení uživatelských jmen přihlášených uživatelů, kteří jsou právě přítomni na dané stránce. K dosažení této funkcionality je potřeba sdílet data mezi procesy. Rozhodl jsem se tedy použít prostředky jazyka PHP pro práci se sdílenou pamětí.

Ve sdílené paměti si ukládám celkem dvě proměnné. Jedna je asociativní pole, kde klíčem je uživatelské jméno a hodnotou je časová značka, kdy jsme naposledy od daného uživatele dostali dotaz. Druhou proměnnou je identifikátor poslední změny, který je při každé změně seznamu inkrementován. Každý klient posílá na server dotaz alespoň každých 45 sekund. Při příjmu tohoto dotazu je do sdílené paměti uložena informace o uživateli, který dotaz poslal (pokud je přihlášen). Pokud uživatel v seznamu ještě není, je uloženo jeho uživatelské jméno spolu s aktuální časovou značkou a je inkrementován

identifikátor poslední změny. Jinak je pouze aktualizována časová značka příslušející k danému uživatelskému jménu.

K distribuci změn seznamu přihlášených uživatelů je opět použita metoda *long polling*. Skript na serveru pravidelně prochází celé asociativní pole uložené ve sdílené paměti a odstraňuje z něj uživatelská jména všech uživatelů, od kterých jsme už déle než 60 sekundy nedostali žádný dotaz. Pokud jsme při této akci smazali některého uživatele ze seznamu, inkrementujeme také identifikátor poslední změny. Klient při svých dotazech posílá identifikátor poslední změny seznamu, kterou má zobrazenou. Pokud není shoda mezi identifikátorem poslaným od klienta a aktuálním identifikátorem, zašleme klientovi celý aktuální seznam přihlášených uživatelů.

Výsledný vzhled celého kreslení na stránkách Frisky je vidět na obrázku 9.3.

Obrázek 9.3: Kreslení se všemi doplňky

Návod na použití knihovny naleznete v příloze G.

9.7 Navrhovaná budoucí vylepší knihovny

- Agregace zpráv. Např. aktualizace kreslení a aktualizace galerie se posílají v samostatných zprávách. Výhodné by bylo spojovat je do jedné zprávy, pokud je to v aktuální situaci možné.
- Podpora třídění obrázků v galerii podle kategorií.
- Odstraňování obrázků z galerie přímo ve webovém rozhraní (v současnosti lze obrázky mazat pouze přímo z databáze).
- Stahování obrázků do galerie v miniaturní velikosti.

10 Závěr

V rámci diplomové práce jsem se nejprve seznámil s technologiemi, které se používají k tvorbě webových stránek. Konkrétně se jedná o jazyky JavaScript (spolu s knihovnou jQuery) a PHP, které byly použity při další práci. Seznámil jsem se také s nástrojem Adobe Flash, který ale nakonec nebyl použit, protože pro zadanou práci byly vhodnější jiné prostředky. V teoretické části práce jsem dále zmapoval oblast bezpečnosti webu. Seznámil jsem čtenáře s různými aspekty bezpečnosti a s hlavními typy útoků.

V první fázi praktické části diplomové práce jsem provedl analýzu webových stránek Frisky. V rámci této analýzy jsem identifikoval nedostatky stránek a poté jsem navrhl jejich řešení. Při následné realizaci byl nejprve vytvořen nový jednotný layout stránek. Dále byla implementována registrace a přihlašování uživatelů a také možnost přesunu obsahu jednotlivých stránek na jiný server. Při implementaci bylo dbáno na pravidla bezpečnosti, tj. aby uživatelé nemohli ve formulářích odeslat data, která by sloužila k napadení serveru nebo jeho ostatních uživatelů.

V další fázi byla provedena analýza hry Puzzle. Byly nalezeny způsoby, jakými bylo možné v této hře podvádět. Na základě analýzy byly následně jednotlivé problémy odstraněny. Dále jsem implementoval rozhraní pro ukládání výsledků a pro získávání výsledků v HTML podobě. Toto rozhraní bylo úspěšně nasazeno.

V poslední fázi praktické části práce jsem navrhl a implementoval knihovnu pro komunikaci uživatelů na webových stránkách. Knihovna obsahuje několik komponent: kreslení, galerii, chat a zobrazení seznamu přihlášených uživatelů přítomných na stránce. Libovolné komponenty knihovny lze nasadit na jakékoliv stránce. Knihovna je nyní nasazena na stránkách Frisky na samostatné stránce, kde umožňuje komunikaci všech návštěvníků.

Všechny požadavky zadavatele byly úspěšně splněny. Rozhraní pro ukládání výsledků her, knihovna pro komunikaci návštěvníků i layout stránek Frisky byly nasazeny. V kapitole 9.7 navrhuji budoucí rozšíření a vylepšení komunikační knihovny. KMA uvažuje, že v budoucnu budou tato rozšíření implementována.

Seznam použitých zkratek

- **AJAX** - Asynchronous JavaScript and XML.
- **API** - Application programming interface.
- **CPU** - Central processing unit.
- **CRC** - Cyclic redundancy check.
- **CSS** - Cascading Style Sheets.
- **DDoS** - Distributed denial-of-service.
- **DOM** - Document Object Model
- **DoS** - Denial-of-service.
- **HTML** - HyperText Markup Language.
- **IIS** - Internet Information Services, dříve Internet Information Server.
- **IP** - Internet Protocol.
- **KMA** - Katedra matematiky.
- **MAC** - Message authentication code.
- **MSN** - The Microsoft Network.
- **PHP** - PHP: Hypertext Preprocessor.
- **SQL** - Structured Query Language.
- **SW** - Software.
- **SWF** - Small Web Format.
- **URL** - Uniform resource locator.

Seznam použité literatury

[1] CHAPMAN, Stephen. *A Brief History of Javascript* [online]. [cit. 15. září 2011]. URL: <http://javascript.about.com/od/reference/a/history.htm>.

[2] *JavaScript Tutorial* [online]. [cit. 2. října 2011]. URL: <http://www.w3schools.com/js/>.

[3] DOYLE, Matt. *Events and Event Handlers* [online]. Zveřejněno 11.12. 2001 [cit. 6. října 2011]. URL: <http://www.elated.com/articles/events-and-event-handlers/>.

[4] *jQuery Tutorial* [online]. [cit. 20. listopadu 2011]. URL: <http://www.w3schools.com/jquery/default.asp>.

[5] *Ajax (programming)* [online]. [cit. 26. listopadu 2011]. URL: <http://en.wikipedia.org/wiki/AJAX>.

[6] *jQuery.ajax()* [online]. [cit. 25. listopadu 2011]. URL: <http://api.jquery.com/jQuery.ajax/>.

[7] *qTip 2 jQuery plugin* [online]. [cit. 21. února 2012]. URL: <http://craigsworks.com/projects/qtip2/docs>.

[8] *History of PHP* [online]. [cit. 20. října 2011]. URL: <http://php.net/manual/en/history.php.php>.

[9] *PHP Tutorial* [online]. [cit. 21. října 2011]. URL: <http://www.w3schools.com/php/>.

[10] GAY, Jonathan. *The History of Flash* [online]. [cit. 23. října 2011]. URL: <http://www.adobe.com/macromedia/events/john_gay/index.html>.

[11] *Adobe Flash* [online]. [cit. 27. října 2011]. URL: <http://en.wikipedia.org/wiki/Adobe_flash>.

[12] DASWANI, Neil - KERN, Christoph - KESAVAN, Anita. *Foundations of Security: What Every Programmer Needs to Know*. 1. vydání. [United States of America]: Apress, 2007. ISBN-13 (pbk): 978-1-59059-784-2.

[13] *How to Find or Validate an Email Address* [online]. Poslední úprava 2. 12. 2010 [cit. 10. února 2012].
URL: <http://www.regular-expressions.info/email.html>.

[14] MALONE, William. *Create a Drawing App with HTML5 Canvas and JavaScript* [online]. [cit. 12. března 2011].
URL:
<http://www.williammalone.com/articles/create-html5-canvas-javascript-drawing-app/>.

Přílohy

A Obsah DVD

Adresářová struktura DVD přiloženého k této práci, vypadá následovně:

- - *doc* - zde naleznete diplomovou práci v elektronické podobě. Adresář *doc* obsahuje tyto podadresáře:
  - *pdf* - obsahuje diplomovou práci ve formátu PDF.
  - *src* - obsahuje zdrojové soubory pro vygenerování práce programem TeX.
- - *src* - zde se nachází programové vybavení vytvořené v rámci této diplomové práce. Obsahuje tyto podadresáře:
  - *lib* - zde je v podadresáři *komunikacni_knihovna* umístěna knihovna pro komunikaci uživatelů webových stránek, viz kapitola 9. V podadresáři *ukladani_vysledku* naleznete programátorské rozhraní pro ukládání herních výsledků a pro získávání nejlepších výsledků v HTML formátu. Více o tomto rozhraní naleznete v kapitole 8.
  - *php* - zde naleznete ukázky nasazení vytvořených programátorských prostředků umístěných ve výše popsaném adresáři *lib*. V podadresáři *drawing* se nachází ukázka nasazení knihovny pro komunikaci uživatelů webových stránek. V podadresáři *Puzzle* naleznete hru, která používá vytvořené rozhraní pro ukládání a zveřejňování výsledků. Tato hra byla také ošetřována proti podvádění, viz kapitola 7.
  - *sql* - zde jsou umístěny SQL skripty pro vytvoření databázových tabulek, které pro své fungování potřebuje knihovna pro komunikaci uživatelů a rozhraní pro ukládání a zveřejňování výsledků her.

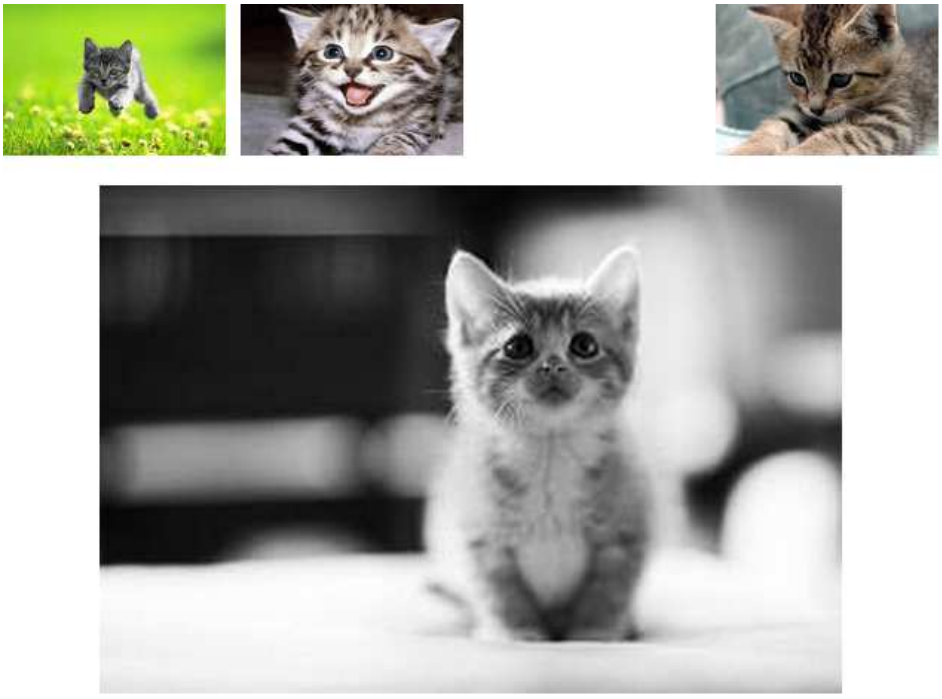
B Galerie - ukázka tvorby animací s knihovnou jQuery

```
jQuery.noConflict();
jQuery(document).ready(function() {
  probihaAnimace = false;
  idZvetsenehoObrazku = "";
  sirkaObrazku = 150;
  vyskaObrazku = 102;
  okrajObrazku = 10
  pocetObrazkuVGalerii = 4;

  sirkaZvetsenehoObrazku = 500;
  vyskaZvetsenehoObrazku = 342;

  poziceZvetsenehoObrazkuX = (pocetObrazkuVGalerii*(sirkaObrazku+
     okrajObrazku)
    -okrajObrazku - sirkaZvetsenehoObrazku)/2;
  poziceZvetsenehoObrazkuY = vyskaObrazku + 20;

  idZvetsenehoObrazku = null;

  jQuery(".kotata").click(function() {
    if(!probihaAnimace) {
      probihaAnimace = true;
      var idObrazku = jQuery(this).attr("id");


      //klepnul-li uživatel na právě zvětšený
      //obrázek, opět jej zmenšíme
      //a vrátíme na původní pozici
      if(idObrazku == idZvetsenehoObrazku) {
        zmensiAVratObrazek(idObrazku, function() {
          idZvetsenehoObrazku = null;
          probihaAnimace = false;
        });
      }

      else {
        //je právě zvětšen jiný obrázek,
        //než který chce uživatel?
```

```
        if(idZvetsenehoObrazku != null) {
          //pokud ano, zmenšeme jej a vrat'me jej
          //na původní pozici
          zmensiAVratObrazek(idZvetsenehoObrazku, function() {
            //pak můžeme vybraný obrázek vyndat
            //z galerie a zvětšit
            premistiAZvetsiObrazek(idObrazku, function() {
              idZvetsenehoObrazku = idObrazku;
              probihaAnimace = false;
            });
          });
        }
        else {
          //žádný obrázek není aktuálně zvětšený
          //mužeme tedy rovnou vybraný obrázek
          //vyndat z galerie a zvětšit
          premistiAZvetsiObrazek(idObrazku, function() {
            idZvetsenehoObrazku = idObrazku;
            probihaAnimace = false;
          });
        }
      }
    }
});

//zmenší obrázek a vrátí jej na jeho původní
//pozici v galerii
function zmensiAVratObrazek(idObrazku, callback) {
  //na čísla v id obrázku určíme jeho původní
  //pozici
  var puvodniPoziceX = (parseInt(idObrazku.substring(4))-1)
   * (sirkaObrazku+okrajObrazku);
  var puvodniPoziceY = 0;

  //zmenšíme obrázek do původní velikosti
  jQuery("#"+idObrazku).animate({width:sirkaObrazku,
   height:vyskaObrazku}, 1000,function() {
    //přesuneme obrázek na jeho původní X pozici
    jQuery("#"+idObrazku).animate({left:puvodniPoziceX}
    , 1000, function() {
      //zasuneme obrázek do galerie
      jQuery("#"+idObrazku).animate({top:puvodniPoziceY}
      , 500, function() {
```

```
            //zde je vidět zavolání callback funkce,
            //pokud byla nějaká zadána
            if(typeof callback == 'function'){
              callback.call(this);
            }
          });
        });
      });
    }

    //vybere obrázek z galerie a zvětší ho
    function premistiAZvetsiObrazek(idObrazku, callback) {
      //vytáhni obrázek z galerie
      jQuery("#"+idObrazku).animate({top:poziceZvetsenehoObrazkuY},
          500,function() {
        //přesuň obrázek na cílovou X pozici tak,
        //aby byl uprostřed galerie
        jQuery("#"+idObrazku).animate({left:poziceZvetsenehoObrazkuX
            }, 1000,function() {
          //zvětši obrázek
          jQuery("#"+idObrazku).animate({width:sirkaZvetsenehoObrazku
              , height:vyskaZvetsenehoObrazku},1000, function() {
            //zde je vidět zavolání callback funkce,
            //pokud byla nějaká zadána
            if(typeof callback == 'function'){
              callback.call(this);
            }
          });
        });
      });
    }
});
```

Vzhled galerie ilustruje Obrázek B.1.



Obrázek B.1: Galerie

C Vzor pro většinu stránek na Frisky

Zde ukáži vzor PHP skriptu, který lze na Frisky použít k vytvoření prázdné stránky o šířce 1020 px, která ale obsahuje všechny prvky layoutu:

```
<? require("../sys/Tfunkce.php"); ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
            "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <link href="../style.css" media="screen" rel="Stylesheet" type="
     text/css" >
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8
     " />
  <script type='text/javascript' src='../sys/jquery.min.js'></
     script>
  <script type='text/javascript' src='../sys/menu.js'></script>
  <script type="text/javascript" src="../jquery.easing.1.3.js"></
     script>

  <script type="text/javascript" src="../sys/Groject.ImageSwitch.
     yui.js"></script>
  <script type="text/javascript" src="../sys/eu_logo.js"></script>
  <script type="text/javascript" src="../sys/loginbox.js"></script>
</head>
<body>
<?php
include '../sys/hlavicka-bez-odkazu.php';
?>
<div id="pagewrapper">
<div id="hlavicka-space"> </div>
<div id="mainpart">
<?php
include '../sys/loginbox.php';
include '../sys/toggleMenu.php';
?>
<div id="content">

</div>
<?php
include '../sys/paticka2.php';
?>
</div> </div>
```

```
</body>
</html>
```

D Skript pro rozšíření layoutu stránek na Frisky

Následující skript se vloží na konec hlavičky stránky na Frisky, která je širší než 1020 pixelů:

```
<script type="text/javascript">
var j = jQuery.noConflict();
j(document).ready(function() {
function resize(element) {
  var contentWidth = element.width();
  var width = contentWidth + j("#hlavicka-space").width();
  j("#pagewrapper").width(width);
  j("#mainpart").width(contentWidth);
  j("#content").width(contentWidth);
  j("#toggleMenu").width(contentWidth);
  j(".menu_wrap").width(contentWidth);
  j(".menu_link").width(contentWidth-10);
  j("#toggleMenu > a").width(contentWidth);
  j("#login").width(contentWidth);
  j("#paticka").width(contentWidth);

  var menuWidth = 1020;

  j("#mainMenuWrapper").css("left", (contentWidth-menuWidth)/2);
}
resize($('#content'));
});
var $ = j.noConflict();
</script>
```

Zde nadefinovaná funkce *resize* přijímá jako parametr jeden HTML element. Tento element musí být vybrán jQuery selektorem. Funkce resize se podívá na šířku daného elementu a poté postupně nastaví všechny prvky layoutu na stejnou šířku. Je-li tedy celý obsah stránky např. uvnitř nějakého elementu DIV, který má šířku větší než 1020, použije se výše uvedený skript tak, že jako parametr funkce resize se vybere právě tento DIV.

Ještě je potřeba vyřešit umístění menu. Jeho šířka je 1020px, u většiny stránek tedy odpovídá přesně šířce obsahu. U širších stránek jsem rozhodl, že menu bude zarovnáno na střed stránky. O to se stará poslední řádka funkce *resize*. Problém ovšem nastává u

stránek, které jsou extrémně široké (některé takové výjimky na Frisky existují). Potom není vhodné menu zarovnat na střed, lepší je ponechat zarovnání na levý okraj stránky. V těchto případech je tedy potřeba smazat poslední řádku funkce *resize*.

E Vzor stránek, jejichž obsah se roztahuje přes okno prohlížeče

HTML kostra těchto stránek vypadá následovně:

```
<?php require("../sys/Tfunkce.php"); ?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
            "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
   >
<title>Frisky</title>
<script type="text/javascript" src="../sys/jquery.min.js"></script>
<link rel="stylesheet" type="text/css" href="../style.css">

<script type='text/javascript' src='../sys/fullscreenmode.js'></
   script>
<script type='text/javascript'>
jQuery(document).ready(function() {
  jQuery(window).resize(function() {
    fullscreen();
  });
});
</script>
</head>
<body>
<?php
include '../sys/hlavicka.php';
?>
<div id="pagewrapper">
<div id="hlavicka-space"> </div>
<div id="content" style="min-height:0px;">

</div> </div>
</body>
</html>
```

Zde se povšimněte, že při změně velikosti okna je volána funkce *fullscreen*. Ta je umístěna v souboru *fullscreenmode.js*, jehož obsah vypadá takto:

```
var j = jQuery.noConflict();

function fullscreen() {
  var hlavicka = j("#hlavicka-space");
  var content = j("#content");

  content.width(j(window).width()-hlavicka.width());
  content.height(j(window).height());
  hlavicka.height(j(window).height());
}

j(document).ready(function() {

  j("#content").css("margin-left", "0px");
  j("#content").css("margin-right", "0px");

  j("#content").css("clear", "none");
  j("#pagewrapper").css("width","100%");
  j("#pagewrapper").css("height","100%");

  fullscreen();
});
```

V levé části okna prohlížeče je vždy hlavička stránky a ve zbytku okna je roztažen element s id *content*, tedy obsah stránky.

F Ukázka použití knihovny pro ukládání a zveřejňování výsledků

Pro použití této knihovny je potřeba mít HTTPD (testováno s Apache httpd 2.2.22 (Win32)) s nainstalovaným PHP5 (testováno s PHP 5.2.10) s knihovnou mysqli. Dále je potřeba mít nainstalovanou databázi MySQL.

Uživatel knihovny by měl nejprve správně nastavit parametry pro přihlášení k databázi na samotném začátku skriptu *scorelib.php*. Tabulku, do které se budou výsledky ukládat, vytvoří uživatel skriptem *create_scores_table.sql*. V tomto souboru je potřeba správně nastavit název databáze.

Použití funkce pro ukládání výsledků by mělo být přímočaré, popis parametrů funkce je uveden v podkapitole 8.2.

Podívejme se jak zveřejňovat výsledky. Nejprve si vytvoříme PHP stript pojmenovaný např. score.php, který bude sloužit k získání tabulky nejlepších výsledků v dané hře. Dejme tomu, že chceme zobrazit výsledky seřazené vzestupně podle dosaženého času a v tabulce chceme tyto dosažené časy zobrazit. PHP skript pak může vypadat následovně:

```
<?php
  include "scorelib.php";

  $idHry = 1;
  $volba;
  if (!IsSet($_GET["volba"])) {
    $volba=5;
  }
  else {
    $volba = $_GET["volba"];
  }

  getBestResults($idHry,$volba,$ORDER_BY_TIME_ASC,$SHOW_TIME);
?>
```

Proměnná *$volba* v příkladu představuje výběr časového období, za které mají být výsledky zobrazeny. Implicitně jsou vybrány nejlepší výsledky všech dob, ve kterých je každý uživatel maximálně jednou. Je vidět, že tento parametr je nastavován parametrem HTTP GET dotazu. Volba časového období zobrazených výsledků je tedy

ponechána uživateli. Ten ji provádí použitím tlačítek, která jsou obsažena v tabulce výsledků.

Podívejme se, jak potom může vypadat HTML dokument, který bude zobrazovat tabulku výsledků poskytnutou výše uvedeným PHP skriptem:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
            "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>Frisky</title>
        <link href="scorelib.css" media="screen"
  rel="Stylesheet" type="text/css" />
  <script src="../sys/jquery.min.js" type="text/javascript">
  </script>
  <script src="scorelib.js" type="text/javascript">
  </script>
  <script type="text/javascript">
    url = "score.php";
    jQuery(document).ready(function(){
      jQuery('#stats').load(url);
    });
  </script>
</head>
<body>
<div id="ScoreBOX" class="ScoreBOX">
<div id='stats' class='stats'></div><div id="tlX" class="tlX">X</
   div>
</div>
</body>
</html>
```

Příklad ukatuje, že do HTML stránky musí být začleněna knihovna jQuery a dále skript *scorelib.js* a CSS soubor *scorelib.css*. Pro správné zobrazení tabulky výsledků je potřeba do naší HTML stránky vložit element DIV s id *ScoreBOX* a také všechny vnořené elementy, které jsou vidět ve výše uvedeném příkladu. Dále by v adresáři *./images/score* měl být umístěn obrázek *scorebox.png*, což je pozadí tabulky výsledků. Pro načtení obsahu tabulky nejlepších výsledků je použita funkce *load* knihovny jQuery, jak je rovněž vidět v příkladu. Uživatel knihovny musí správně nastavit hodnotu proměnné *url* na URL PHP skriptu, který nám poskytne tabulku výsledků. V našem případě se jedná o PHP skript *score.php*, jehož příklad jsem ukázal výše. Proměnná *url* je velmi důležitá, neboť s její existencí a správným nastavením počítá i samotná knihovna (skript v souboru *scorelib.js*).

G Návod na použití knihovny pro komunikaci uživatelů

Pro použití je potřeba mít HTTPD (testováno s Apache httpd 2.2.22 (Win32)) s nainstalovaným PHP5 (testováno s PHP 5.2.10) s knihovnou mysqli. Dále je potřeba mít nainstalovanou databázi MySQL.

Knihovna pro komunikaci uživatelů obsahuje následující adresáře a soubory:

- Adresář *commlib* - obsahuje PHP skripty knihovny: *functions.php*, *getImage.php*, *getOlderImages.php*, *getUpdates.php*, *manageRequests.php*, *saveChat.php*, *saveImages.php*, *saveImagesUpdate.php*, *stopUpdate.php*.
- Adresář *css* - obsahuje CSS styly: *commlib.css*, *jquery.qtip.min.css*.
- Adresář *js* - obsahuje javascripty: *commlib.js*, *jquery.ajaxmanager.js*, *jquery.qtip.min.js*.
- Soubor *backend.php* - tento PHP skript je také součástí knihovny, ale uživatel si v něm musí upravit hodnoty proměnných podle toho, jak chce mít knihovnu nastavenou.

Dále knihovna obsahuje tyto SQL skripty:

- create_drawings_table.sql - slouží k vytvoření databázové tabulky pro ukládání čar nakreslených uživateli na canvas.
- create_chat_table.sql - vytvoří tabulku pro ukládání zpráv, které uživatelé napsali do chatu.
- create_images_table.sql - vytvoří tabulku pro ukládání obrázků nakreslených na canvas, které pak mají být vystaveny v galerii.

Uživatel knihovny musí správně nastavit název databáze na první řádce každého z těchto skriptů. Dále si může změnit názvy tabulek vytvářených v uvedených SQL skriptech. Knihovna předpokládá použití databáze MySQL. Tabulky musí být kódované v UTF-8.

Když máme vytvořeny potřebné databázové tabulky, musí uživatel provést nastavení knihovny v souboru *backend.php*. Všechny AJAX požadavky budou posílány na tento skript. Následuje ukázka tohoto skriptu s komentáři vysvětlujícími význam jednotlivých proměnných, jejichž hodnotu je potřeba nastavit:

```
//Nastavení ID stránky. Je důležité proto, aby
//nedocházelo ke konfliktům proměnných
//uložených v sezení, pokud by byla knihovna
//nasazena na více stránkách ve stejné doméně.
$pageID = "test";

//Zde uživatel vybere, které části knihovny chce
//na své stránce použít.
//Použít kreslení?
$useDrawing = true;
//Použít galerii?
$useGallery = true;
//Použít chat?
$useChat = true;
//Použít zobrazení přihlášených uživatelů?
//Nelze použít pod OS Windows.
$useLoggedUsers = true;

//Nastavení parametrů připojení k MySQL databázi.
//hostname nebo ip adresa databáze
$DB_HOST = "localhost";
//MySQL uživatelské jméno
$DB_USERNAME = "test";
//heslo
$DB_PASSWORD = "test";
//název databáze
$DB_DBNAME = "test";

//Nastavení názvů databázových tabulek
//Název tabulky pro uložení čar.
$drawingsTableName = "drawings";
//Název tabulky pro uložení chatu.
$chatTableName = "drawing_chat";
//Název tabulky pro uložení obrázků galerie.
$imagesTableName = "drawing_savedImages";

//Jak často se má skript dotazovat DB,
//zda je v ní nová aktualizace.
//Jednotkou jsou milisekundy.
$updateCheckPeriodMs = 200;
//Po jaké době nejpozději skript skončí
//a klient pak musí poslat nový dotaz.
//Jednotkou jsou sekundy.
```

```
$scriptTimeoutSecs = 45;

//Má knihovna logovat svojí činnost?
//Logování se provádí PHP funkcí error_log.
$log = false;

/*Nastavení kreslení*/
//Celková délka čar (v počtu bodů, jejichž
//spojením jsou složeny), které mají být
//poslány uživateli při příchodu na stránku.
$maxLength = 15000;

/*Nastavení galerie*/
//Kolik obrázků má být načteno do galerie
//po příchodu na stránku.
$maxGalleryVisibleImages = 12;

/*Nastavení chatu*/
//Kolik posledních zpráv má být maximálně
//zobrazeno v chatu po příchodu na
//stránku.
$maxChatHistoryMsgs = 10;
//Jak daleko do minulosti má sahat historie
//zpráv, které jsou zobrazeny po příchodu na stránku.
$chatHistorySince = time() - 60*60;

/*Nastavení sdílené paměti pro seznam přihlášených uživatelů.
Sdílenou paměť ani funkci ftok nelze použít pod OS Windows*/
//Velikost sdílené paměti v bytech.
$sharedMemSize = 5000000;
//Klíč (identifikátor) semaforu.
$semkey = 1;
//Klíč (identifikátor) sdílené paměti.
$shmkey = ftok("/var/www/FRISKY/drawing/drawingshm","d");

//Cesta k adresáři, ve kterém jsou umístěny
//PHP skripty knihovny.
$commLibDir = "commlib";

require($commLibDir."/manageRequests.php");
```

Dále musí uživatel knihovny napsat svůj skript v jazyce JavaScript, ve kterém zavolá funkci *init* komunikační knihovny. Tato funkce musí být zavolána uvnitř jQuery *ready*

bloku, např:

```
jQuery(document).ready(function(){
  init("backend.php");
});
```

Funkce *init* provede počáteční nastavení knihovny, vytvoří potřebné HTML elementy a nakonec zavolá funkci, která provede počáteční načtení dat ze serveru a následně stahuje všechny další aktualizace ze serveru. Funkce *init* přijímá tyto parametry:

- *backendUrl* - URL souboru *backend.php*, viz výše. Pokud chce uživatel použít pouze chat a/nebo zobrazení seznamu přítomných přihlášených uživatelů, potom stačí zadat pouze tento parametr.
- *drawingWidth* - šířka kreslícího plátna v počtu pixelů.
- *drawingHeight* - výška kreslícího plátna v počtu pixelů.
- *color* - barva kreslené čáry.
- *lineWidth* - šířka kreslené čáry.
- *galleryWidth* - šířka galerie v počtu obrázků. (Pozn.: Tento a další parametry musí uživatel vyplnit pouze pokud chce s kreslením používat i galerii.)
- *galleryHeight* - výška galerie v počtu obrázků.
- *imagesWidth* - šířka obrázků v galerii v počtu pixelů.
- *imagesHeight* - výška obrázků v galerii v počtu pixelů.

Pokud by chtěl uživatel na stránku umístit galerii bez použití kreslení, dosáhne toho takto:

```
jQuery(document).ready(function(){
  //šířka galerie
  gWidth = 5;
  //výška galerie
  gHeight = 3;
  //šířka obrázků
  iWidth = 100;
  //výška obrázků
  iHeight = 66;
  init("backend.php");
});
```

Pro změnu barvy kreslené čáry musí uživatel knihovny svým javascriptovým programem měnit proměnnou *curColor*. Hodnotou této proměnné musí být řetězec, který začíná křížkem (#) následovaným šesti hexadecimálními číslicemi, které definují RGB reprezentaci barvy. Pro změnu šířky kreslené čáry je potřeba měnit hodnotu proměnné *curSize*. Hodnotou musí být celé číslo.

Dále je potřeba upravit hlavní PHP skript, na který uživatelé posílají dotaz při příchodu na stránku. Do stránky musíme vložit CSS styly *commlib.css* a *jquery.qtip.min.css*. Dále musí být vložena knihovna jQuery a tyto javascriptové soubory: *jquery.qtip.min.js*, *jquery.ajaxmanager.js* a *commlib.js*. Dále musíme do PHP skriptu vložit tyto řádky:

```
session_start();
$pageID = "test";
$_SESSION["active".$pageID] = true;
```

session_start samozřejmě voláme pouze pokud jsme tuto funkci nevolali dříve. Proměnná *$pageID* musí být nastavena na stejnou hodnotu, jako stejně pojmenovaná proměnná ve skriptu *backend.php*. Proměnnou v sezení zde nastavujeme proto, aby mohla knihovna za určitých okolností rychleji detekovat, že uživatel odešel ze stránky.

Uživatel knihovny musí také do stránky vložit některé HTML elementy podle toho, které prvky knihovny chce použít:

- **kreslení** - Musí být do stránky vložen DIV element s id *drawingDiv*. Pro mazání canvasu je potřeba ještě vložit element s id *clearCanvas*. Po klepnutí myší na něj se smaže obsah canvasu.

- **galerie** - Do stránky je potřeba vložit DIV element s id *gallery*. Pro uložení obrázku nakresleného na canvasu do galerie je nutné vložit na stránku také tlačítko s id *submitImage*. Potom by se také mělo na stránce nacházet textové políčko (element *input* typu *text*)s id *imageDesc*, kam mohou uživatelé napsat popis obrázku, který bude uložen spolu s obrázkem.

- **chat** - Pro chat je potřeba do stránky vložit celkem tři elementy. Do elementu DIV s id *chatarea* budou vkládány všechny příspěvky v chatu. Dále musí být na stránce element TEXTAREA s id *chatinput*, do kterého budou moci uživatelé psát svoje příspěvky. Nakonec je potřeba do stránky umístit tlačítko s id *sendMsg*. Po klepnutí na něj se odešle zpráva napsaná v textovém poli *chatinput*. Nadefinování kaskádových stylů těchto elementů je ponecháno uživateli knihovny. U elementu *chatarea* doporučuji nastavit parametr *overflow* na *auto*, aby příliš dlouhé řetězce bez bílých znaků nemohly "přetéct"daný element. Aby javascriptový program uměl správně vypsat do chatu uživatelské jméno místního uživatele

(pokud je přihlášen), je potřeba umístit do stránky další element. Pokud je uživatel přihlášen přes systém Orion, musí být na stránce element s id *usernameOrion*, jehož obsahem bude dané uživatelské jméno. V případě přihlášení přes místní registraci musí být uživatelské jméno obsahem elementu s id *usernameLocal*.

- **přihlášení uživatelé** - Pro výpis seznamu přihlášených uživatelů je potřeba do stránky umístit DIV element s id *users*. Do něj bude tento seznam vypisován. Nastavení stylu tohoto elementu je opět ponecháno uživateli knihovny.