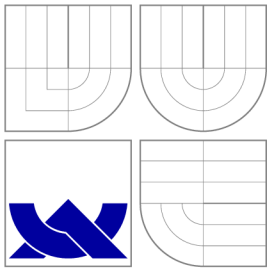
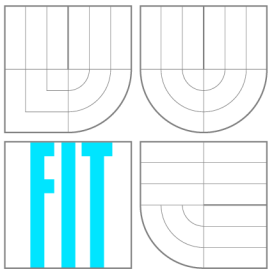


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁSTROJ PRO ANALÝZU PSANÍ UŽIVATELE NA KLÁVESNICI JAKO DOPLNĚK PRO FIREFOX

A TOOL FOR ANALYSIS OF USER'S TYPING SKILLS AS AN ADD-ON FOR FIREFOX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ TUREK
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2012

Abstrakt

Cílem této bakalářské práce je vytvořit aplikaci, která běží na pozadí a ukládá statistická data o psaní na klávesnici. Na vyžádání uživatele se data v přehledné podobě vyhodnotí. Tato zpětná vazba má pomoci ke zdokonalení techniky psaní na klávesnici. Implementačně je aplikace realizována jako doplněk pro Firefox. V této technické zprávě se zabývám teorií hmatové metody ovládání klávesnice deseti prsty, objasňuji motivaci pro vytvoření mé aplikace a popisuji její návrh a implementaci. Samostatná část je věnována technologiím, které jsou specifické pro vytváření doplňků pro Firefox. Na závěr se pokouším na základě zkušeností od uživatelů objektivně zhodnotit přínos této práce.

Abstract

The objective of this Bachelor thesis is to create an application running in background, saving statistic data about typing. This data is evaluated on user's demand and displayed in a transparent form. The purpose of this feedback is to help improve typing skills. The aplication is implemented as a Firefox add-on. In this technical report I focus on the theory of touch typing technique, clarify the motivation to create my application and describe its design and implementation. There is a separate section devoted to technologies that are specific to developing Firefox add-ons. In the summary I objectively try to evaluate the benefits of this work based on experience from the users.

Klíčová slova

psaní na klávesnici, hmatová technika psaní, generování statistik, doplněk pro Firefox, XUL, CoffeeScript

Keywords

typing, touch typing technique, generating statistics, Firefox add-on, XUL, CoffeeScript

Citace

Lukáš Turek: Nástroj pro analýzu psaní uživatele na klávesnici jako doplněk pro Firefox, bakalářská práce, Brno, FIT VUT v Brně, 2012

Nástroj pro analýzu psaní uživatele na klávesnici jako doplněk pro Firefox

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Turek
14. května 2012

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu mé bakalářské práce, panu doc. Ing. Adamovi Heroutovi, Ph.D., za konstruktivní kritiku, návrhy na zlepšení a vstřícnost.

Obsah

Kapitola 1

Úvod

Elektronická forma komunikace se již dávno stala běžnou součástí každodenního života. I když se stále více rozšiřují technologie pro přenos hlasu po internetu, nemůžeme písemné komunikaci odepřít její důležitost. Z hlediska úspory času je vhodné psát co nejefektivněji, tedy rychle a bez chyb. Výukových programů psaní na klávesnici všemi deseti prsty existuje celá řada. Cílem této práce však není provázet uživatele lekcemi psaní, ale sledovat jeho úroveň psaní při běžné komunikaci a prezentovat mu výsledky v přehledné podobě, která pro něj bude motivační zpětnou vazbou pro individuální trénink.

Mohlo by se zdát, že internetový prohlížeč je určen jen ke čtení obsahu, a mnoho lidí jej skutečně používá pouze k načítání webových stránek. Přesto se do něj občas nějaký text zadává, minimálně při vyplňování URL v adresním řádku, odesílání formulářů apod. Jsou i tací, kteří prohlížeč používají k psaní e-mailů, chatování či vytváření blogů. Analýza psaní v internetovém prohlížeči tudíž má své opodstatnění a stává se podnětem pro vznik nového doplňku Firefoxu.

V následujícím textu naleznete teorii správných návyků při psaní všemi deseti prsty, dozvíte se, komu a proč může být výsledná aplikace užitečná, a pozornost bude věnována i existujícím alternativám. Samostatná kapitola je vyčleněna pro obecný popis tvorby doplňku pro Firefox, a to od počátků vývoje až po hledání cest k jeho zviditelnění na internetu. Po tomto teoretickém úvodu přichází část zabývající se návrhem architektury aplikace. Zde podrobně rozvádím nároky na výsledný program a analyzuji možná rizika. Následuje popis implementace, rozbor jednotlivých funkčních celků a zmiňuji také problémy, které musely být při realizaci řešeny. Na základě reakcí uživatelů a návrhů na vylepšení shrnuji dosavadní přínos tohoto projektu a nastiňuji směr, jakým by se mohl dále vyvíjet.

Kapitola 2

Hmatová technika psaní na klávesnici

Hmatovou technikou psaní rozumíme způsob ovládání klávesnice všemi deseti prsty rukou, kdy text píšeme bez sledování klávesnice, aniž bychom přemýšleli nad tím, kde se která klávesa nachází. Tato část publikace je úvodem do správného psaní touto technikou, budou vysvětleny základní pojmy a ukázána cesta ke zlepšování výkonů.

2.1 Jak ovládnout klávesnici

K výuce psaní můžeme použít tištěné průvodce, kde nácvik provádíme v textovém editoru opisováním jednotlivých cvičení z knihy, např. titul Deseti prsty na klávesnici [9] nebo anglická učebnice Touch typing in 10 minutes [3]. Existuje však celá řada softwarových alternativ, z nichž je pro českou klávesnici jedním z metodicky nejpropracovanějších výukový program ZAV [12], jehož autorem je Jaroslav Zaviačič, vicemistr světa v psaní na stroji z roku 1967. V této souvislosti bych rád upozornil na laciné výukové softwary vyvíjené byť zdatnými programátory, kteří ovšem nedisponují znalostmi správné a efektivní výuky psaní.

Základní poloha pro obě ruce je na prostřední řadě písmen, kde prsty levé ruky jsou položeny od ukazováku po malík na klávesách „F, D, S, A“ a prsty pravé ruky od ukazováku po malík na klávesách „J,K,L,ů“, viz. [9]. Palce se opírají o mezerník. Ostatní znaky jsou prstům přiděleny podle toho, který je k nim vzdáleností nejbližší. Při jejich psaní se prst přesouvá ze základní polohy na jejich místo a vrací se zase zpět.

Pro úspěšné zvládnutí techniky psaní na klávesnici všemi deseti prsty je důležité zautomatizovat si jednotlivé pohyby, nikoli pamatovat si pozici jednotlivých kláves a k nim přidružených znaků. V opačném případě dochází k poklesu soustředěnosti na obsah textu, protože je mozek zaměstnán řízením uvědomělých pohybů před stisknutím každé klávesy. Ve chvíli, kdy toto začneme dělat podvědomě, budeme jednotlivá slova vnímat jako celek a nikoli po jednotlivých znacích. To samozřejmě vyžaduje chvíli cviku a trpělivosti. Zdroj [10] uvádí, že minimální doba pro zvládnutí hmatové techniky pro opis textu (tedy bez speciálních znaků jako @,$,*) je tři měsíce, včetně těchto znaků pak pět měsíců.

Uvedené zdroje, ze kterých jsem čerpal, se zabývají i jinými správnými návyky při psaní jako je správná výška židle, neopírání zápěstí o klávesnici, vhodný úhel loktů od těla, vzdálenost od monitoru apod., avšak podrobnější rozbor těchto aspektů by vybočoval od hlavního tématu této publikace.

2.2 Výhody proti psaní se sledováním klávesnice

Největším přínosem a zároveň i častou motivací pro učení hmatové techniky je nepochybně úspora času. Jak bylo zmíněno výše, jedním z benefitů je i lepší soustředěnost na obsah textu. Není výjimkou, že člověk při hledání konkrétního písmena na klávesnici zapomene, co chtěl vlastně napsat. K nárůstu efektivity dochází i z hlediska opravování chyb. U zažitých stereotypních pohybů prstů si totiž často uvědomíme překlep, aniž bychom viděli výsledek na obrazovce.

Jiným úhlem pohledu jsou ergonomické aspekty. Rovnoměrné rozložení zátěže na jednotlivé prsty omezuje únavu drobných svalů. Pan Neugebauer ve své knize [10] mimo jiné uvádí, že z ergonomického hlediska je vhodnější střídání palců mačkajících mezerník v závislosti na předchozím znaku, kdežto pro sportovní výkony v psaní na klávesnici je efektivnější ovládání mezerníku jen jedním palcem podle toho, zda je uživatel levák či pravák, což vede ke zvyšování rychlosti psaní.

Při opisu textu z předlohy hmatovou technikou šetříme také oči, protože nedochází k přeostřování zraku mezi předlohou, monitorem a klávesnicí. Zároveň můžeme zamezit vytáčení krční páteře, pokud máme předlohu přímo před sebou a monitor umístíme do strany.

2.3 Sledované atributy během psaní

Klíčovou vlastností pro hodnocení výkonnosti při psaní je rychlost. Ta se udává v hrubých, resp. čistých, úhozech za minutu. Počtem hrubých úhozů rozumíme všechny znaky, které byly napsány. Tedy ačkoli například pro napsání znaku „Ť“ je potřeba stisku čtyř kláves (levý shift, háček, pravý shift, T), počítá se jako jeden úhoz, viz. také [10]. Počet čistých úhozů získáme po odečtení neopravených chyb od hrubých úhozů. U soutěžních disciplín v psaní na počítači bývá zpravidla za neopravenou chybu strháváno více úhozů.

Tím se dostáváme k dalšímu důležitému prvku při hodnocení, kterým je chybovost. Nejčastěji bývá vyjádřena jako procentuální podíl chybných znaků na celkovém počtu úhozů. Nemá smysl hnát se za vyšší rychlostí psaní na úkor velké chybovosti, a proto je prioritní soustředit se na psaní bez chyb. Chybovost je velmi negativně hodnocena i na zmiňovaných soutěžích, a proto je běžné, že pomalejší soutěžící s precizní technikou psaní poráží rychlejšího s větším počtem chyb. Při tréninku bychom se měli zaměřovat na naše slabá místa a procvičovat nejvíce slova obsahující znaky nebo kombinace znaků, v nichž děláme nejčastěji chyby.

Zejména v začátcích při postupném osvojovování si hmatové techniky je důležité myslet na plynulost psaní a snažit se psát rytmicky. To můžeme zpětně kontrolovat například histogramem odstupů mezi jednotlivými úhozy.

2.4 Zvyšování rychlosti se zkratkovými systémy

Hbitost prstů nelze zvyšovat do nekonečna a i při intenzivním tréninku se tak časem dostaneme do fáze, kdy počet našich úhozů přestane narůstat. Existuje však alternativní metoda psaní pomocí tzv. zkratkových systémů, se kterou můžeme tuto hranici překonat. Její princip je podobný těsnopisu a spočívá v tom, že místo jednotlivých slov či slovních spojení píšeme pouze zkratky. Tyto zkratky se v programu pro editaci textu nahrazují automaticky zpět jejich původním významem. Velmi dobře propracovaným českým zkratkovým

systémem je ZAVpis [12], který vytvořil již výše zmíněný Jaroslav Zaviačič. V soutěžních disciplínách může být použití ZAVpisu velkou výhodou, vyžaduje to ovšem měsíce tréninku a učení se zkratek. Pro běžnou elektronickou komunikaci jako je psaní e-mailů je to však zbytečná komplikace.

2.5 Rozložení znaků

Mluvíme-li o hmatové metodě psaní, je v této souvislosti důležitým pojmem rozložení znaků na klávesnici. To definuje zastoupení znaků jednotlivými klávesami. Nejtypičtějším rozložením je QWERTY, které bylo v Česku prohozením písmen „Z“ a „Y“ transformováno na QWERTZ. Tyto standardy vznikaly v době vývoje prvních psacích strojů a obsahují definici základních kláves. V současné době se pro psaní na počítači používají různé národní standardy včetně speciálních znaků, které z těchto základní typů vycházejí.

Byly zaznamenány i snahy prosadit ve světě jiná rozložení místo nejpoužívanějšího QWERTY. Rozložení DVORAK má nejčastěji používaná písmena v angličtině umístěna v základní poloze, čímž lze efektivitu při psaní zvýšit přibližně o 12%, viz. [10]. V praxi se ale s tímto rozložením kláves příliš často nesetkáme.

Kapitola 3

Tvorba doplňku pro internetový prohlížeč Firefox

Internetový prohlížeč Mozilla Firefox je dílem společnosti Mozilla Corporation ve spolupráci s externími programátory z řad dobrovolníků, kteří jsou do projektu zapojeni díky jeho otevřenému kódu. Od své první verze uvedené na trh v roce 2004 prošel rozsáhlým vývojem a postupně si získal přízeň uživatelů po celém světě. Jeho předností proti tehdy nejrozšířenějšímu prohlížeči Internet Explorer byla vyšší bezpečnost, rychlost a jednoduchost. Zároveň je ale jednoduchost Firefoxu náročnějším uživatelům kompenzována množstvím doplňků, které si mohou nainstalovat. Tato kapitola je stručným průvodcem tvorbou těchto doplňků a na jejím konci by měl mít čtenář představu o technologiích a principech, které jsou při tom uplatněny. Rozbor jednotlivých programovacích jazyků by byl nad rámec této publikace.

3.1 Doplněk, rozšíření nebo zásuvný modul?

V první fázi mé bakalářské práce jsem se svým vedoucím při upřesňování zadání hovořil o „pluginu“ pro Firefox. Při studiu potřebných podkladů jsem však zjistil, že Mozilla zastřešující tento internetový prohlížeč rozlišuje mezi pojmy doplněk (angl. addon), rozšíření (angl. extension) a zásuvný modul (angl. plugin). Dovolil bych si proto nejdříve ujasnit používané názvosloví.

Doplněk zde figuruje jako termín nadřazený rozšíření a zásuvnému modulu, je to tedy obecnější pojem. Rozšířením se rozumí instalační balíček, který Firefox obohacuje o nové funkce. Tvůrci tohoto internetového prohlížeče se snaží o jeho jednoduchost a právě díky rozšířením si jej uživatelé mohou přizpůsobit vlastním potřebám. Zásuvný modul je naproti tomu software třetích stran manipulující s obsahem, nejčastěji pro zpřístupnění videa, animací a her [11]. Příkladem zásuvného modulu je Windows Media Player, Java nebo Adobe Reader.

Z výše uvedeného vyplývá, že při použití správné terminologie lze můj cíl formulovat jako vytvoření rozšíření pro Firefox, které lze obecně nazvat doplňkem.

3.2 Používané technologie

Firefox jako webový prohlížeč je sám o sobě realizován s použitím webových technologií a to nám také určuje nástroje potřebné pro tvorbu doplňku – XUL, XPCOM, JavaScript a

CSS. Z pohledu webového vývojáře se tedy vedle známého skriptovacího jazyka JavaScript a jazyka kaskádových stylů CSS objevují nové pojmy XUL a XPCOM.

XUL je jazyk pro tvorbu uživatelského rozhraní založený na XML, principiálně je velmi podobný značkovacímu jazyku HTML pro tvorbu webových stránek. XPCOM je multiplatformní model komponent používaný v aplikacích Mozilly. Díky rozhraní, které poskytuje například pro práci se soubory, pamětí nebo řetězci, můžeme vytvářet opravdu komplexní platformě nezávislé doplňky [11].

3.3 Protokol pro adresování komponent

Produkty Mozilla jsou koncipovány tak, aby všechny jejich součásti byly adresovatelné pomocí URL. K tomuto účelu slouží protokol `chrome` a vzniká tak speciální tvar URL adresy, jejíž obecný formát je `chrome://balíček/část/soubor`. U doplňku se za „balíček“ dosadí jeho vlastní název. Nutno dodat, že tímto způsobem přistupujeme nejčastěji k souborům typu XUL, ale může se jednat o soubor v libovolném formátu. Lze tedy použít absolutní cestu i k souborům JavaScriptu, definicím stylu, obrázkům, apod.

Na kód načtený do prohlížeče protokolem chrome se narozdíl od klasického webového obsahu nevztahují žádná omezení. Je tak nasnadě zanést si do vyvíjeného doplňku bezpečnostní díru načítáním vzdáleného obsahu. Ten lze totiž podvrhnout a injektovat tak škodlivý kód uživateli, viz. [7]. Na tento fakt není příliš upozorňováno, a tak jsem si tyto dopady plně uvědomil až při návrhu, viz. kapitola 4.9.

3.4 Integrace doplňku do Firefoxu

Celý postup, který se v následujícím textu snažím srozumitelně shrnout s přidáním vlastních zkušeností, je velmi dobře popsán v elektronické knize od Jamese Edwarda [4]. Já se zde zaměřuji na základní principy, s jejichž pochopením by měl čtenář získat základní přehled o způsobu, jakým se doplňky do Firefoxu přidávají.

Z hlediska přehlednosti je dobré dodržovat adresářovou strukturu znázorněnou na obrázku 3.1 v podkapitole Kompletace instalačního balíčku. Závazné jsou ale pouze názvy souborů `install.rdf` a `chrome.manifest`, které jsou klíčové pro instalaci. K ostatnímu obsahu jsou právě v `chrome.manifest` uvedeny cesty. V tomto výkladu se však budu držet obecných zvyklostí, a proto pojmenujeme složku s hlavním obsahem `content`. Ta je jádrem celé aplikace a slouží jako kořenový adresář pro naše zdrojové kódy. Složka `skin` by měla být úložištěm definic stylů CSS a složku `locale` zavádíme v případě potřeby více jazykových verzí.

3.4.1 Vytvoření ovládacího prvku

Když vyvíjíme doplněk pro Firefox, budeme chtít pravděpodobně nějakým způsobem zpřístupnit uživatelské rozhraní přímo z prohlížeče, například přidáním položky v menu nebo tlačítka na lištu. Pro tento účel potřebujeme vytvořit XUL soubor, ve kterém definujeme umístění nového prvku do současné podoby prohlížeče, tzv. „overlay“ (v překladu „překrytí“). Toto označení dobře vystihuje i způsob práce se souborem.

Nejdříve otevřeme již existující XUL soubor s definicí hlavního okna, který lze nalézt na adrese `chrome://browser/content/browser.xul`, a jeho editací zjistíme název prvku, kam chceme naši položku umístit (např. `toolbarpalette` pro nástrojovou lištu). Kvůli lepší orientaci je vhodné použití nástroje DOM Inspector, jenž při procházení zdrojového

textu graficky zvýrazňuje přidruženou oblast okna, viz. [11]. Element, ve kterém chceme položku vytvářet, vložíme do nově vytvořeného souboru `overlay.xul` a jeho vnitřní prvky nahradíme vlastním, např. `toolbarbutton`. Zbývá ještě přidat akci, která se provede po kliknutí na prvek. To realizujeme nastavením atributu `command` s voláním příslušné funkce.

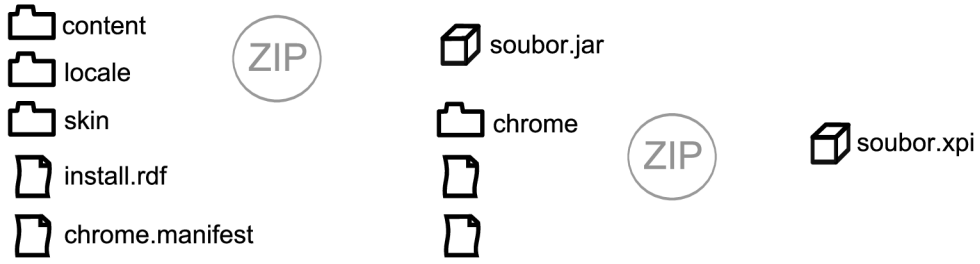
3.4.2 Rozšíření o více jazykových verzí

Za účelem podpory multijazykových doplňků zavádí Mozilla balíček „locale“ [4]. Pro každý jazyk vytvoříme složku s definičním DTD souborem, který bude obsahovat entity pro jednotlivá slova či sousloví. Princip rozlišení řetězců podle lokalizace spočívá v odkazování se na tyto entity z XUL souboru. Do chrome.manifest přidáme informaci o cestě k DTD souboru pro každý námi definovaný jazyk a na základě aktuálního jazyku uživatelského rozhraní Firefoxu se automaticky vybere patřičný DTD soubor.

Někdy potřebujeme rozlišit podle lokalizace i řetězce v JavaScriptu, např. v textu dialogových oken. To lze provést s použitím rozhraní XPCOM vytvořením odpovídající komponenty. Následným voláním funkce `GetStringFromName()` získáme řetězec pro stávající lokalizaci, čímž dosáhneme stejného efektu jako referenčním odkazem v XUL souborech.

3.4.3 Kompletace instalačního balíčku

V konečné podobě je rozšíření pro Firefox jedním souborem s příponou XPI. Jeho přetažením myší do okna Firefoxu se automaticky objeví výzva k zahájení instalace. Tvorba instalačního balíčku je složena z pěti kroků popsaných níže, viz. také [11], a je znázorněna na následujícím obrázku.



Obrázek 3.1: Kroky vedoucí k vytvoření instalačního balíčku

1. Vytvoříme prázdnou složku `chrome`
2. Složky `content`, `locale` a `skin` zabalíme do archivu ZIP, jehož koncovku přepíšeme na JAR a umístíme do složky `chrome`
3. Zálohujeme původní soubor `chrome.manifest`
4. Obsah souboru `chrome.manifest` upravíme tak, aby cesty k jednotlivým komponentám byly ve tvaru `jar:chrome/náš_soubor.jar!/komponenta/`
5. `chrome.manifest`, `install.rdf` a složku `chrome` zabalíme do archivu ZIP a koncovku přepíšeme na XPI

Z uvedeného postupu je patrné, že dekompozicí kopírující tento návod v opačném pořadí získáme zpět zdrojový kód. Všechna rozšíření pro Firefox jsou tedy v principu tzv. „open source“ (otevřený zdrojový kód). Pro znečitelnění kódu je možno použít obfuskátor, ze strany uživatele to ovšem nepůsobí příliš důvěryhodně.

3.5 Možnosti publikace a automatické aktualizace

Po dokončení vývoje doplňku přichází fáze rozšíření mezi koncové uživatele. Pomineme-li variantu aktivního obesílání potenciálních uživatelů, máme dvě základní možnosti, jak své dílo publikovat. Můžeme využít webové stránky Mozilly, kde lze zveřejnit vlastní doplněk a zařadit jej tak na seznam oficiálních doplňků Firefoxu. Pokud z nějakého důvodu chceme dostupnost omezit jen pro určitou skupinu lidí (například u projektů se specifickým účelem, které pro široké spektrum běžných uživatelů nemají využití), budeme poskytovat stažení z vlastních webových stránek.

3.5.1 Hostování na oficiálním serveru Mozilly

U univerzálních doplňků se předpokládá snaha autora získat co nejvíce uživatelů a oficiální publikace je bezesporu prvním správným krokem k tomuto cíli. V porovnání například s osobním webem vývojáře je faktor vysoké návštěvnosti webu Mozilly umocněn důvěrou uživatele v ověřený zdroj.

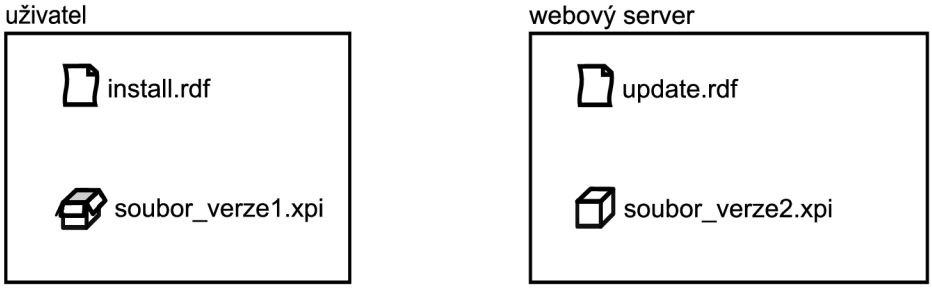
Před umístěním na seznam oficiálních doplňků Firefoxu se provádí jeho kontrola. Lze volit mezi zařazením k předběžné nebo plné kontrole. U předběžné kontroly není ověřována funkčnost, doplněk je označen jako experimentální, s varováním o možném poškození počítače. Uvádí se [6], že tento typ kontroly je proveden do tří dnů od nahrání souboru. Plná kontrola se zaměřuje i na funkčnost a zdrojové kódy, přičemž uváděná doba čekání je do deseti dnů. Z vlastní zkušenosti ale dodávám, že u obou těchto alternativ je v praxi čekání ve frontě o mnoho delší, a to až v násobcích oficiálního limitu.

Výhodou této cesty ke zveřejnění doplňku je samozřejmě větší publicita, ale zároveň i jednoduchost automatických aktualizací, které se po schválení nové verze provádí u všech uživatelů, aniž by se autor musel o cokoliv starat. Velmi užitečné je také rozhraní pro správu verzí souborů a zobrazení statistik o frekvenci stahování a počtu aktivních uživatelů.

3.5.2 Správa na vlastním webovém serveru

U druhého přístupu k publikování doplňků, tedy umístěním na vlastní webové stránky, se musíme o automatické aktualizace postarat sami vytvořením souboru `update.rdf`, který odkazuje na nejnovější instalační soubor XPI. Zároveň musí být v `install.rdf` uvedena cesta k aktualizačnímu souboru, viz. [8]. Tuto závislost jsem graficky znázornil na obrázku 3.2. Instalační i aktualizační soubor obsahuje číslo verze XPI souboru, jehož je součástí, resp. na který odkazuje, a pokud je při porovnání těchto hodnot na straně uživatele zjištěna existence novější verze, dochází k automatickému stažení a instalaci XPI souboru z webového serveru.

K zajištění funkčnosti výše popsaného konceptu je potřeba provádět aktualizace přes zabezpečený protokol HTTPS nebo vytvořený doplněk digitálně podepsat. Zatímco provoz HTTPS je podmíněný zakoupením certifikátu, nástroj pro digitální podpis doplňku lze použít zdarma.



Obrázek 3.2: Reference mezi soubory při aktualizaci s použitím vlastního webového serveru

Mozilla pro tyto účely poskytuje nástroj McCoy, který vygeneruje veřejný a soukromý klíč. Veřejný klíč se vloží do souboru `install.rdf` přidáním elementu `<em:updateKey>` a soukromým klíčem se podepíše soubor `update.rdf`. V rámci kompatibility musíme vyplnit v obou těchto souborech shodné ID doplňku, stejně tak minimální a maximální podporovanou verzi prohlížeče Firefox. Hvězdička zde ovšem nemá význam jakékoli verze, což by nás mohlo zmást kvůli její sémantice v regulárních výrazech, ale zastupuje nejvyšší verzi. Proto se doporučuje [8] používat ji pouze pro maximální podporovanou verzi.

Poslední akcí, kterou před nahráním doplňku na web musíme provést, je kontrola souboru `.htaccess` a případně přidání pravidel pro práci s XPI a RDF soubory. Jedná se o jednoduché dva řádky, ale bez jejich přítomnosti by tyto typy souborů nebyly rozpoznány, čímž pádem by nefungovala instalace doplňku z webu ani automatické aktualizace. Jejich podoba je následující:

```
AddType application/x-xpinstall .xpi
AddType text/xml .rdf
```

Nyní je vše připraveno pro publikaci doplňku na našich webových stránkách. Výhodou správy souborů na vlastním serveru je nezávislost na Mozille a možnost uvolnění nové verze ihned po jejím dokončení bez čekání na proces kontroly. Na druhou stranu právě kvůli absenci této kontroly může být snížena důvěryhodnost ze strany uživatelů, kteří při stahování doplňků z alternativních zdrojů nemají žádnou záruku bezpečnosti.

Kapitola 4

Návrh aplikace pro analýzu psaní

K otázce návrhu přistupuji se stanovením cílů, kterých chci dosáhnout, a proto nejprve hledám stávající produkty v této oblasti a posléze představuji hlavní myšlenky mé práce. Jádrem této kapitoly je popis základních stavebních prvků a postupů použitých při návrhu a dozvíte se také o technologických omezeních, se kterými jsem musel počítat.

Výsledný produkt by měl být určen širokému spektru uživatelů, v zásadě komukoli, kdo má zájem sledovat své schopnosti při psaní. Více se chci ovšem zaměřit na uživatele píšící všemi deseti prsty, kteří na svém výkonu chtějí pracovat a tento nástroj by jim to měl umožnit.

4.1 Co nového by měl nástroj nabídnout?

Hlavní motivací pro vznik této práce bylo vyplnit mezeru v současném spektru aplikací orientujících se na psaní na klávesnici, a tak mými prvními kroky bylo studium existujících alternativ k tomuto tématu. Překvapivě jsem zjistil, že již samotný nápad sledovat psaní uživatele při běžném používání osobního počítače, je poměrně originální, neboť stávající aplikace se vydávají spíše směrem aktivní výuky. Jedná se sice také o vyhodnocování statistik psaní, ale na základě opisu textu ze zadané předlohy. Konkrétně v oblasti doplňků pro Firefox je v tomto duchu navržena aplikace Addictive Typing Lessons.

Programem, který má blíže k zadání mé práce, je Typing Speed Monitor realizovaný jako rozšíření prohlížeče Google Chrome. Tento nástroj jsem stáhl a otestoval. Zjistil jsem, že jeho jádrem je vizualizace klávesnice a barvou jednotlivých kláves je znázorněna rychlost jejich stisku. Při najetí myší na jednotlivé klávesy je pak zobrazena informace o celkovém počtu stisknutí a průměrné rychlosti. Údaje se zobrazují za celou dobu používání programu. Nevýhodou pro českého uživatele je absence rozložení klávesnice QWERTZ, protože v nastavení lze volit jen mezi rozloženími QWERTY, Dvorak a Coleman.

Od zmíněných výukových programů je odlišnost mého konceptu patrná. – Neanalyzuje se opisovaný text, ale libovolný vstup od uživatele, který zadává z klávesnice při práci ve Firefoxu. Pozornost uživatele přitom směřuje pouze k jeho hlavní činnosti, protože monitorování probíhá na pozadí a bez interakce s uživatelem. Rozdíl je také ve způsobu zobrazení výsledků. Vyhodnocení se provede jen na vyžádání uživatele a po dopsání textu není obtěžován tabulkou se statistikami, ale může si získané hodnoty kdykoli zpětně zobrazit a sledovat tak dlouhodobý vývoj.

Chci se ale vydat i jiným směrem ve srovnání s rozšířením Typing Speed Monitor pro Google Chrome, které jsem popisoval výše. Mým cílem není vygenerovat pouze souhrnná

data získaná za celou dobu používání, ale poskytnout uživateli přehled o jeho dlouhodobém vývoji. Zároveň chci upozorňovat na špatné návyky a časté chyby. Zatímco tato konkurenční aplikace pohlíží na backspace jako na běžnou klávesu, já se chci více zaměřit na její používání a především znaky, které jsou s ní opravovány. Co se týče rozložení klávesnice, plánuji podporu QWERTZ i QWERTY s možností výběru v mé aplikaci.

4.2 Ochrana soukromí jako priorita

Přesvědčit potenciální uživatele o čistých úmyslech tohoto projektu se zdá být nelehkým úkolem, obzvláště když si většina pod „analýzou psaní“ představí klasický špionážní program, jehož jediným smyslem je zaznamenávat stisky kláves do souboru, který je následně zneužit pro získání citlivých dat. Úplné zamezení pochyb je asi nemožné, abych však alespoň částečně předešel spekulacím, uvádím zde zásady ochrany soukromí, které při návrhu plně ctím a respektuji.

To nejpodstatnější je, že text, který uživatel napíše, se nikam neukládá. Z nasbíraných dat se získají pouze statistické údaje, které sledujeme, ale samotný obsah se zahazuje. Jediné, co z původního obsahu zůstane, jsou znaky opravené klávesou backspace, přičemž při mazání více znaků po sobě se vybírá jen poslední smazaný. Zrekonstruovat na základě těchto údajů původní text je tedy zcela nemožné. V původním konceptu jsem počítal i se zobrazováním celých slov s nejčastějšími chybami, kde by se kvůli ochraně hesel prováděla kontrola shody se slovníkem. Kvůli vysokým nárokům na výkon jsem však od tohoto nápadu nakonec upustil.

Druhou charakteristikou ochrany soukromí je ukládání veškerých dat lokálně. Bez vědomí uživatele neprobíhá žádný přenos po internetu. Zde jsem zvažoval i možnost dobrovolného zapojení se do srovnání výsledků s ostatními uživateli, ale nový prvek soutěživosti by mohl zastínit hlavní smysl projektu, tedy osobní rozvoj uživatele v technice psaní. Jediným spojením s okolním světem je v mém návrhu formulář pro odeslání zpětné vazby.

4.3 Hledání cesty k uživateli

Když jsem stál před návrhem této aplikace, bylo pro mě zásadní otázkou, jakým směrem se vydat, aby nástroj co nejlépe plnil svůj účel a byl pro uživatele opravdu užitečný. Zamýšlel jsem se nad tím, jaká data budou relevantní pro sběr a jakým způsobem je prezentovat. Hledání cesty k uživateli tedy představuje nalezení odpovědi na otázku, jak vytvořit tuto aplikaci, aby uživatele motivovala k lepším výkonům.

Protože základními parametry pro hodnocení zručnosti při psaní na klávesnici jsou rychlost a chybovost (viz. kapitola 2.3 „Sledované atributy během psaní“), je vyhodnocování těchto údajů jádrem mého návrhu. Určení rychlosti je jednoznačné na základě četnosti zaznamenaných úhozů. Problém nastává v případě chybovosti, jelikož nemáme žádný referenční text jako předlohu k porovnání. I kdyby se každé slovo porovnávalo na shodu se slovníkem, vznikne mnoho nejednoznačností, nemluvě o psaní bez diakritiky či v cizím jazyce. Proto se v návrhu nesnažím postihnout chyby, které zůstaly neopravené, ale zaměřuji se naopak na opravování klávesou backspace. Vycházím z faktu, že časté vracení se v textu snižuje efektivitu psaní a upozorním-li uživatele na nadměrné používání backspace s dodatečnou informací o problémových znacích, dám mu tím impulz k dosažení vyšší přesnosti.

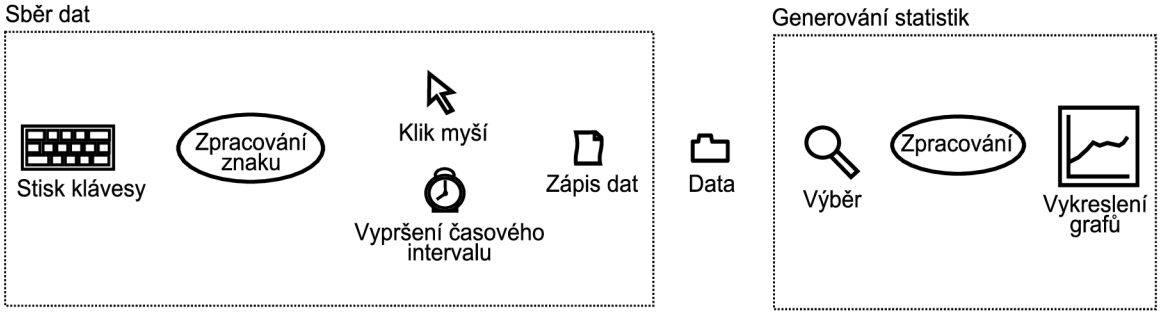
Při volbě způsobu generování statistik jsem dospěl k názoru, že největší vypovídající hodnotu budou mít data vynesená do grafu. V úvahu připadalo i slovní hodnocení, ale

kromě toho, že by se jednalo o mou interpretaci toho, co lze vyčíst i z těchto grafů, by mohl dlouhý text odrazovat od čtení. Sledováním dlouhodobého vývoje je jasně vidět, jestli se výkonnost zlepšuje či naopak zhoršuje, ale poznatkem u konkrétního člověka může být například i to, že v dopoledních hodinách píše lépe než večer.

Často opomíjenou složkou hodnocení hmatové techniky je rytmičnost. Správným návykem při psaní je mezi klávesami přecházet plynule, a proto jsem se rozhodl zaznamenávat časové odstupy mezi úhozy, které se následně vyhodnotí v podobě histogramu. Ten bude představovat součet všech dat za období zvolené uživatelem.

4.4 Architektura aplikace

Základní principy mého návrhu ilustruji na obrázku 4.1. Celek jsem rozdělil na dva samostatné funkční bloky – sběr dat a generování statistik. Dá se říci, že se vzájemně doplňují, neboť první z nich čeká na vstup uživatele a zapisuje data a druhý tato data čte a poskytuje výstup pro uživatele. Zásadní rozdíl je v tom, že první část je aktivní po celou dobu otevřeného okna prohlížeče, zatímco druhá část se spouští na povel uživatele. Pro názornost je schéma zjednodušené a podrobnějšímu popisu obou bloků se věnuji v dalším textu.



Obrázek 4.1: Základní návrh aplikace

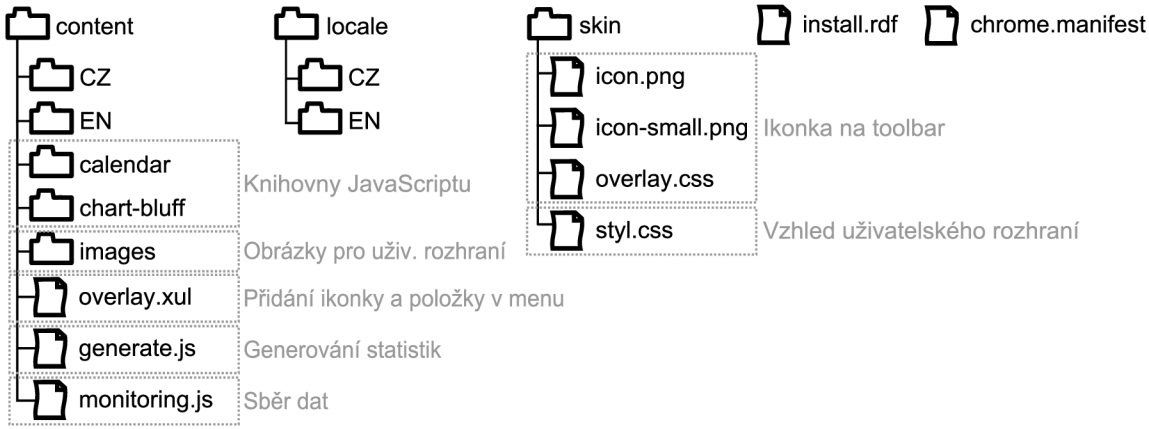
Jak ukazuje schéma, spouštěcím faktorem pro zahájení sběru dat je první stisk klávesy. Poté se vyhodnocuje každý další úhoz zvlášť. Je potřeba určit, kde skončilo psaní souvislého textu, aby se o něm mohl vytvořit záznam. Prvním případem, kdy považuji psaní textu za ukončené, je detekce kliknutí myší. Protiargumentem může být, že se uživatel takto může pouze vrátit na jiné místo v textu a pokračovat v psaní dále, ovšem z hlediska monitorování se stejně touto akcí ztratí kontext a bylo by obtížné získat přehled o tom, ve které části již napsaného textu se zrovna kurzor pohybuje. Proto pro mě znamená každé kliknutí myší vyhodnocení napsaného textu (pokud nějaký existuje). Protože je cílovým produktem doplněk internetového prohlížeče, je častým případem klikání přechod mezi políčky formuláře nebo mezi adresním řádkem a hlavní částí okna. Druhou možností pro ukončení souvislého textu, kterou beru v potaz, je vypršení předem daného časového intervalu, který uplyne od posledního stisku klávesy. Experimentálně jsem optimální hodnotu pro tento interval stanovil jako 5 sekund. K delší odmlce při psaní totiž dochází jen při větším zamyšlení. Dlouhé čekání je nežádoucí i z pohledu hodnocení rychlosti, protože by tato odmlka po zprůměrování zkreslovala celkový výsledek.

Před ukládáním dat do souboru se nejdříve vyhodnocuje, jestli byl text dostatečně dlouhý. Toto jsem pro zachování jednoduchosti do obrázku nezahrnul. Například osamocený stisk jedné klávesy nemá žádnou vypovídající hodnotu, a proto se v takovém případě

nebude vytvářet nový záznam. Můj původní návrh dále počítal s kategorizací textu do skupin „krátký“, „středně dlouhý“ a „dlouhý“ na základě doby jeho psaní. Tento nápad byl později nahrazen selekcí rozmezí délky textů, jež mají být započítány do statistik, v uživatelském rozhraní. Rozdělení do zmíněných skupin je subjektivní záležitostí a nalézt všeobecně vyhovující hranici není lehké. Takto si tedy každý může filtrovat texty konkrétní délky, které ho zajímají.

4.5 Adresářová struktura

Než se pustím do detailnějšího popisu jednotlivých částí, představím rozmístění souborů a jejich funkci, kterou mají plnit. Schematicky jsem adresářovou strukturu znázornil na obrázku 4.2. Tento celek bude na závěr zabalen do souboru XPI podle pravidel uvedených v části 3.4.3 „Kompletace instalačního balíčku“



Obrázek 4.2: Adresářová struktura

První úroveň jsem pro úsporu místa zobrazil horizontálně. Jedná se o instalační soubory a tři složky pro hlavní obsah, jazykové balíčky a vzhled. Na první pohled je patrný dvojí výskyt složek pro češtinu a angličtinu. Není to však duplikát v návrhu nebo chyba obrázku, protože na obou místech mají tyto složky trochu jiný význam. V adresáři `content` tvoří jejich obsah jazykové verze HTML souborů a bude se mezi nimi přepínat na základě volby uživatele, kdežto v `locale` bude obsahem definiční DTD soubor odkazovaný z `overlay.xul` a složka se vybere automaticky podle jazykového prostředí prohlížeče. Pomocí `locale` bude definován pouze text pro spouštěcí položku v menu.

Funkční část návrhu aplikace představují soubory `monitoring.js` pro sběr dat a `generate.js` pro generování statistik, tzn. dva bloky, na které jsem aplikaci rozdělil v předchozím textu o architektuře návrhu. Kromě mých vlastních zdrojových kódů se zde objevují knihovny, z nichž první bude sloužit k práci s daty formou grafického kalendáře a druhá k vykreslování grafů.

Kostrou uživatelského rozhraní jsou HTML soubory v podsložkách `CZ` a `EN` složky `content`. Vzhled je definován v souboru `styl.css` a obrázky se načítají se složky `images`. Předefinování nástrojové lišty a umístění položky v menu je obsahem `overlay.xul` a ze složky `skin` se bude načítat ikonka.

4.6 Sběr dat

Tato část aplikace nemá uživatelské rozhraní a běží pouze na pozadí. Její úlohou je shromažďovat statistická data, která budou později využita k reprodukci pro uživatele. Forma ukládání dat je volena s ohledem na efektivní využití úložného prostoru, a proto se snažím předcházet jejich duplikaci, jak bude dále ukázáno na příkladu histogramu. Ačkoli se v porovnání s generováním statistik jedná o méně rozsáhlou část, bude muset být při implementaci laděna se zvlášť velkou pozorností, protože kód programu je načítán spolu s otevřením okna prohlížeče a chyba by zde mohla znamenat pád Firefoxu při běžném používání, v nejhorším případě hned po spuštění.

4.6.1 Zpracování znaku

Nyní se dostávám k odhalení černé skříňky, kterou na obrázku 4.1 představuje proces zpracování znaku. Zde se nejprve kontroluje, jestli se jedná o alfanumerický znak, neboť funkční a řídicí klávesy nepředstavují součást textu. Výjimkou je klávesa backspace, která je naopak zajímavá z hlediska opravování chyb, a proto se bude také zpracovávat.

Spolu se začátkem nového textu se spustí časovač, který bude hlídat uplynulou dobu od posledního stisku klávesy a zároveň bude počítadlem celkové doby psaní. U každého dalšího zachyceného znaku se naopak bude zkoumat časový interval mezi tímto a předchozím stiskem klávesy a tato informace se uloží pro pozdější tvorbu histogramu.

Další akcí následující po stisku klávesy je uložení příslušného znaku. Tato informace je udržována pro účely sledování opravovaných znaků. Řetězec tvořený několika posledními znaky napsaného textu totiž můžeme při detekci klávesy backspace zpětně procházet a zjistit tím, jaký byl poslední smazaný znak a jakým byl nahrazen. Důležitou součástí návrhu je také počítadlo celkově napsaných znaků, které se s každým alfanumerickým znakem inkrementuje a s klávesou backspace naopak dekrementuje.

4.6.2 Vyhodnocení textu

Fázi zápisu do souboru předchází vyhodnocení nasbíraných dat. Jak jsem již naznačil, příliš krátký text není statisticky zajímavý, a proto jsem zvolil spodní hranici pro akceptování textu tři sekundy, aby mohly být sledovány například krátké odpovědi v chatovacím okně, ale zároveň eliminovány zprávy typu „OK“.

Počet úhozů se vydělí délkou psaní v minutách, čímž získáme rychlost v úhozech za minutu. Z toho vyplývá, že u záznamů kratších než minuta se dělí zlomkem menším než jedna. Výsledným efektem je umělé dopočítání rychlosti u krátkých textů a tudíž často i zkreslení rychlosti. Z toho důvodu jsem váhal, zda zahrnovat tyto krátké texty do statistik rychlosti. Nakonec jsem je ponechal s tím, že je může filtrovat uživatel při výběru sledovaných hodnot.

Před zápisem dat je dále potřeba připravit data pro histogram, poněvadž ukládání rychlosti stisku klávesy pro každý jednotlivý úhoz by vedlo ke zbytečně velkému objemu dat. Získané údaje o časových intervalech mezi úhozy se navzorkují po pěti milisekundách (hodnota odvozena empiricky na základě hrubosti výsledného histogramu) a následně se budou hledat výskyty stejných hodnot, čímž vzniknou dvojice „čas, počet výskytů“.

Zbylé údaje není třeba před zápisem zpracovávat, neboť byly průběžně vytvářeny v požadované podobě již během psaní. Jedná se o začátek psaní textu, celkový čas psaní a opravované znaky.

4.6.3 Forma ukládání dat

Data se budou ukládat ve formátu XML do vytvořené složky TypingStats umístěné v profilové složce prohlížeče. Zamýšlel jsem se nad tím, že by nebylo vhodné mít jen jeden datový soubor, protože u každého čtení a zápisu by po delší době používání programu rostly paměťové nároky, a proto jsem se rozhodl vytvářet jeden soubor denně. Bude se tedy vyhledávat soubor s názvem obsahující aktuální datum ve formátu „yyyy_mm_dd.xml“ a v případě jeho neexistence se tento vytvoří. V původním plánu jsem navíc uchovával tyto názvy v souboru s metadaty. Vzhledem k tomu, že jsem později objevil způsob, jak získat názvy všech souborů ve složce s rozhraním XPCOM, ukázalo se toto řešení jako zbytečná komplikace.

S každým novým záznamem se v kořenovém prvku XML souboru vytvoří potomek, jehož vnořené elementy odpovídají sledovaným atributům při psaní, tedy čas začátku, doba psaní, úhozy za minutu, data histogramu a opravené chyby (původní a nahrazený znak).

4.7 Generování statistik

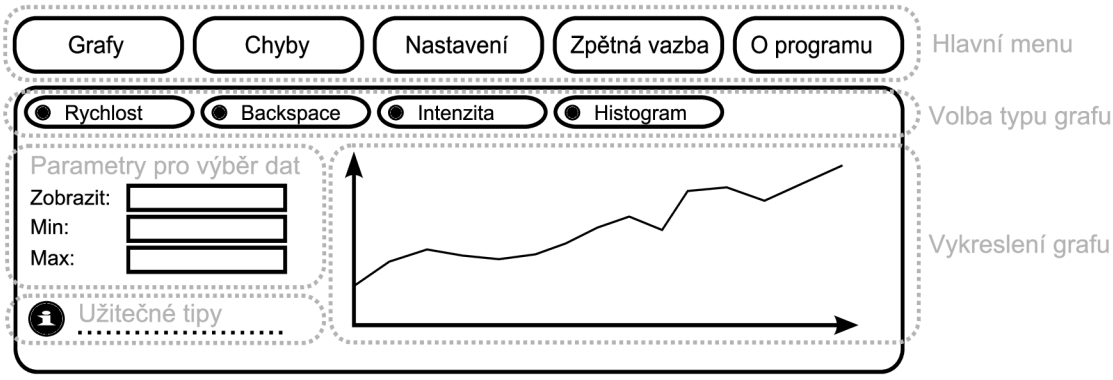
Druhou samostatnou částí vedle sběru dat je zpracování a prezentování statistik. Tento modul je však mnohem více komplexní vzhledem k tomu, že se stará o vyhodnocení všech nasbíraných dat a zároveň grafickým výstupem vytváří zpětnou vazbu pro uživatele. Při jeho popisu postupuji dekompozičně od základních principů až k detailům řešených při návrhu. První, co na uživateli zanechá dojem, je grafické rozhraní, a proto začínám právě u jeho vlastností.

4.7.1 Uživatelské rozhraní

Mou vizí o uživatelském rozhraní bylo již od počátku dialogové okno vyvolané kliknutím na ikonu na liště. Rozhodl jsem se přidat ještě jeden ovládací prvek s toutéž funkcí do záložky „Nástroje“ v menu. Okno se tedy otevře buď kliknutím na obrázkovou ikonu nebo volbou položky „Analýza psaní“ v menu. Měl jsem v úmyslu použít pro generování obsahu jazyk XUL jako standardní jazyk pro tvorbu uživatelských rozhraní ve Firefoxu, ale kvůli kompatibilitě s ostatními knihovnami jsem se nakonec uchýlil k jazyku HTML, více viz. kapitola „Realizace výsledné aplikace“.

Rozvržení prvků se podobá klasické webové stránce s horizontálním členěním, kde nahoře je logo a pod ním menu následované samotným obsahem. Hlavní částí a zároveň úvodní obrazovkou bude vykreslování grafů. Má představa je taková, aby si uživatel v submenu zvolil typ grafu, který ho bude zajímat, na levé straně zvolil parametry pro filtrování dat a vpravo se vykreslil výsledný graf. Nastavitelnými parametry by přitom mělo být sledované období a výběr délky textů, které se budou do statistik zahrnovat. Sledované období bude možno vybírat z přednastavitelných hodnot (např. poslední týden), ale nechávám i možnost volby vlastního intervalu. V takovém případě se meze intervalu volí ze zobrazeného kalendáře výběrem konkrétního dne. Pro lepší představu jsem můj koncept rozmístění prvků načrtnul na obrázku 4.3.

Všechny nastavitelné údaje budou perzistentní, čehož chci docílit ukládáním aktuální konfigurace, která se bude přepisovat při každé změně a načítat spolu s otevřením sekce grafů. Pokud by tedy chtěl konkrétní uživatel sledovat například vývoj chybovosti v posledních sedmi dnech s omezením na texty delší než minutu, nemusí tyto atributy znovu vybírat po každém spuštění.



Obrázek 4.3: Rozvržení uživatelského rozhraní

Další sekcí bude hodnocení chybovosti na základě používání klávesy backspace. Zde chci upozorňovat na nejčastěji opravované znaky vypsáním deseti nejfrekventovanějších kombinací „původní znak → nahrazený znak“ po řádcích včetně jejich četnosti. Poměrně experimentální formou prezentace opravovaných znaků je grafické znázornění chybovosti jednotlivých prstů, kde uvedená hodnota v procentech u konkrétního prstu bude znamenat jeho podíl na celkové chybovosti.

V sekci „Nastavení“ umožňuji smazání všech nasbíraných dat a přepínání jazyka uživatelského rozhraní mezi češtinou a angličtinou. Při prvním spuštění se vybere automaticky podle jazykového prostředí Firefoxu (čeština pro české prostředí, angličtina pro jakékoli jiné) a dále bude vždy zachován jazyk, který si uživatel nastavil. Balíček „locale“ standardně řešící multijazyčnost nelze použít pro HTML kód, takže budu muset udržovat HTML soubory pro každý jazyk zvlášť.

Sousloví „zpětná vazba“ v tomto textu často používám pro označení motivování uživatele na základě jeho výsledků. V kontextu hlavního menu, tak jak je uvedeno na obrázku 4.3, raději upozorňuji na opačný význam. Míněna je v tomto případě reakce uživatele na moji aplikaci odesláním vyplněného formuláře.

4.7.2 Výběr dat pro graf

Po spuštění uživatelského rozhraní se načtou všechny soubory XML a následně se z nich vyberou potřebná data. Vybraný obsah bude záležet na zvolených parametrech. Ve výchozím nastavením se zobrazí první typ grafu, čili rychost, v období posledních 24 hodin bez omezení délky textu. Při změně těchto hodnot se nové nastavení použije místo výchozího při další inicializaci. I během přepínání mezi jednotlivými typy grafů jsem se rozhodl zanechat ostatní parametry nezměněny, aby bylo možné srovnání mezi různými veličinami za stejné období.

První hodnocenou vlastností je rychlost v čistých úhozech za minutu, čili bez přepisovaných znaků. Jak jsem již vysvětlil dříve, není možné bez známé předlohy jednoznačně určovat neopravené chyby, a tudíž se čistým úhozům v mém pojetí programu rovná počet znaků textu, který se zobrazí jako výsledek. Každý záznam o psaném textu v sobě uchovává informaci o počtu těchto úhozů a chceme sledovat průměr těchto hodnot. To znamená, že pro každý bod v grafu bude jeho hodnota výpočítána jako aritmetický průměr všech zaznamenaných rychlostí v dané hodině nebo dni.

Druhým typem grafu je chybovost, kterou jsem v menu označil raději přesnějším ná-

zvem „backspace“. I když se chyby u opisovaného textu standardně hodnotí procentuálním podílem na celkovém počtu znaků, porovnáním grafů jsem zjistil, že pro mé účely má větší vypovídající hodnotu, když budu udávat častost používání backspace namísto srovnávání s původní délkou textu. Proces výběru dat probíhá podobným způsobem jako u rychlosti s tím rozdílem, že se nejdříve musí spočítat výskyt opravovaných znaků a teprve pak se přejde k aritmetickému průměru.

Intenzita představuje celkový čas strávený psaním textu a tuto statistiku jsem mezi grafy zařadil hlavně kvůli povědomí uživatele o době, kterou věnuje psaní v prohlížeči. Předchozí dva typy grafů se totiž opírají o průměrné hodnoty a chyběla by informace o součtu času. Díky tomu například člověk může zjistit, že při intenzivním psaní musí častěji opravovat znaky a že by si měl dělat více přestávek. Lze tušit, že rozdíl ve zpracování dat ve srovnání s předchozími grafy bude právě ve sčítání místo průměrování, o čemž bude informován také uživatel v oblasti pro užitečné tipy.

Histogram je grafem zcela odlišným už ve své podstatě, protože nevyjadřuje průběh, ale distribuci dat. Kdežto předchozí tři statistiky jsem prezentoval spojnicovým grafem, tento bude sloupcový. Jak jsem již zmínil, využitím má být kontrola plynulosti psaní na základě naměřených intervalů mezi stisky kláves. Vstupní data představují celkovou četnost každé jednotlivé hodnoty a získají se postupným součtem pro každý bod histogramu zvlášť, přičemž se opět vybírá jen z odpovídajících záznamů. Uživatel si nastaví jemu vyhovující rozsah definičního oboru na 0 – 500 ms nebo 0 – 1000 ms. Přepnutím se přepočítá rozlišení histogramu na menší či větší.

4.7.3 Zobrazení grafu

Pro vykreslování grafů jsem si vybral knihovnu Bluff, která je open source a narozdíl od placených alternativ je i jednodušší, a tedy datově méně objemná. Způsobu práce s ní se věnuji v kapitole 5 o implementaci, ale už ve fázi návrhu jsem musel přemýšlet o jejích možnostech a rozhraní.

Zjistil jsem, že předávanou hodnotou budou pouze data, zatímco popisky časové osy si budu muset vytvořit sám. Zvažoval jsem možnost manuální volby rozlišení grafu po hodinách, dnech, atp., avšak zatížení procesoru při vykreslování velkého počtu bodů již bylo znát, a tak jsem se rozhodl řídit toto rozlišení samotnou aplikací na základě počtu zobrazených dní. Hranice jsem volil empricky podle hustoty bodů v grafu. Jako optimální se ukázalo dělit po hodinách intervaly v maximální délce do tří dnů, po dnech do 365 dnů, po měsících do 5000 dnů a větší intervaly (což je už spíše hypotetické) po letech. Podle tohoto pravidla se po zvolení intervalu určí základní časová jednotka, z ní se odvodí potřebný počet bodů, spočítají se hodnoty pro každý bod a vygenerují se popisky časové osy.

V souvislosti s tímto jsem se zabýval i vhodnou hustotou popisků grafu, protože snaha o uvádení všech časových značek vedla k jejich vzájemnému překrývání. Proto budu frekvenci opakování popisků počítat vzorcem $f \doteq \frac{pocet\_bodu}{c}$, kde konstantu c jsem experimentováním navrhl pro každou používanou jednotku času zvlášť.

4.7.4 Hodnocení chyb

Samostatnou částí je vedle statistických dat vynesených do grafu také hodnocení opravovaných chyb. O průběžné četnosti je uživatel informován grafem s názvem „backspace“, zde je však smyslem vytvořit jakýsi jeho profil zachycující slabá místa. Z dat získaných od uživatele chci tedy udělat souhrn toho, co mu dělá nejvíce problémy. Výsledky jsem se rozhodl prezentovat dvěma způsoby – výpisem nejopravovanějších znaků a chybovostí prstů.

Hledání nejčastěji opravovaných znaků probíhá tak, že se z uložených dat posbírají všechny dvojice „původní znak → nahrazený znak“, seřadí se podle četnosti a zobrazí se deset nejfrekventovanějších. Tím dobře vynikne častá záměna konkrétního znaku za jiný. Pro názornost bude v těchto dvojicích barevně odlišen původní (chybný) znak červenou a nahrazený (správný) znak zelenou barvou. Tzv. „bílé znaky“ (mezera, enter) se budou muset vypisovat názvem, aby nevznikalo prázdné místo. Protože se jedná o seznam výsledných znaků, nikoli kláves, není tento výpis nijak závislý na konkrétním rozložení klávesnice.

Nezávislost na rozložení klávesnice naopak neplatí u hodnocení chybovosti prstů. Zde jsou relevantní výsledky podmíněny jednak použitým rozložením QWERTY nebo QWERTZ a zároveň správným rozložením prstů na klávesnici. Používané rozložení nelze JavaScriptem detekovat, proto v sekci „nastavení“ dávám na výběr, podle jakého rozložení se má hodnocení provádět. Rozdíl při špatném nastavení ale nebude nikterak dramatický, neboť problémové jsou jen písmena "Z", "S"a speciální znaky, které jsou v běžném textu málo používané. Moje myšlenka rozložení chybovosti na jednotlivé prsty vychází z toho, že při psaní všemi deseti prsty má každá klávesa jednoznačně přidělen prst, kterým se mačká. Na opravený znak pak pohlížím tak, že tento prst nebyl na svém místě, a tudíž mu připočítávám chybu. Porovnáním s celkovým počtem chyb pak určuji jeho podíl na chybovosti. Z toho jasně vyplývá, že například u psaní dvěma prsty nemá žádný význam tuto statistiku sledovat.

4.8 Omezení při návrhu

Během návrhu této aplikace jsem bohužel narazil na některé překážky, které změnily mou původní vizi o konečné podobě. Jedná se zejména o dříve plánovanou funkčnost, jejíž realizaci brání možnosti JavaScriptu. V mnohém mi pomohlo použití rozhraní XPCOM, ale ani to, jak se ukázalo, není všemocné. Rád bych nyní zmínil několik původních záměrů, jež nakonec kvůli technologickým omezením nebyly realizovány.

Měl jsem představu o zavedení sekce „Správné návyky“, která by sledovala dodržování zásad správného psaní hmatovou technikou na základě toho, co lze vyčíst jen ze stisku kláves. Jednalo by se o vybrané úkony, jejichž správné a špatné provádění by bylo hodnoceno procentuálním podílem. Hlavní měřenou vlastností mělo být používání protilehlé klávesy shift při psaní velkých písmen. Tzn., že pokud se daný znak píše levou rukou, měl by se psát spolu s pravým shiftem a naopak. Problémem je, že JavaScript neumí detekovat konkrétní klávesu shift. Lze pouze zjistit, že byl shift stisknut, ale nikoli s informací, zda se jednalo o levý či pravý.

Jinou vlastností, která měla být sledována, bylo správné psaní číslic. U výskytu čísel v běžném textu není žádoucí používání numerické klávesnice, poněvadž přechody mezi ní a běžným rozložením prstů zdržují. S ovládáním stylu psaní čísel hmatovou metodou má mnoho lidí problémy, ale ve výsledku lze takto při psaní ušetřit mnoho času. I zde jsem narazil na omezení JavaScriptu, který sice umí detekovat stisk klávesy numlock, ale nikoli její stav „on/off“. Řešení mého nápadu sledováním shiftu by nikam nevedlo, protože s přepnutím na anglickou klávesnici lze čísla psát i bez něj.

Chtěl jsem ještě hlídat psaní velkých písmen bez přepínání klávesy Caps Lock, což nakonec zůstalo jako jediný proveditelný nápad, ale vytváření samostatné sekce kvůli této jediné vlastnosti jsem shledal zbytečným. Poznamenejme ještě, že kdybychom mohli uživatele sledovat i vizuálně, daly by se hodnotit i atributy jako správné držení těla, sledování obrazovky bez dívání se na ruce nebo třeba dodržování správného rozložení prstů na klávesnici. To je však daleko nad rámec toho, k čemu jsem při návrhu směřoval.

4.9 Formulář pro zpětnou vazbu a bezpečnostní rizika

Pro zjednodušení komunikace mezi uživateli a mnou jsem se rozhodl vytvořit formulář, jehož prostřednictvím bude možné odesílat připomínky a návrhy k tomuto programu. V případě, že by byl někdo skeptický k odesílání dat z formuláře, uvádím pod ním i kontaktní e-mailovou adresu. Výsledný efekt bude stejný, protože vyplněnou zprávu zpracuje externí PHP skript, který ji odesílá na tento e-mail. Jedná se tedy o jedinou část aplikace, která se připojuje k internetu. Přijaté zprávy budou tříděny podle zvoleného typu na obecnou zpětnou vazbu, dotaz a nahlášení chyby v programu.

Protože je v mém zájmu získat od uživatelů jejich názor na přínos aplikace, vybízím je k vyplnění a odeslání tohoto formuláře po čtrnácti dnech používání. Po spuštění se zkontroluje, jestli zpětná vazba ještě nebyla vyplněna a zda jsou uložena data alespoň za 14 dní. V takovém případě se objeví v informační oblasti prosba o vyplnění a odeslání dosavadních zkušeností s tímto programem.

Prvoplánově jsem celý formulář hodlal implementovat na vzdáleném serveru, odkud by se načítal. Později jsem zjistil, že by to bylo hrubou chybou, protože načítáním externích souborů JavaScriptu bych v mém programu vytvořil bezpečnostní díru, viz. [7]. Vykonávaný kód v doplňcích Firefoxu s oprávněním „chrome“ není nijak omezován a podvržením vzdáleného obsahu by útočník získal přístup k souborům apod. S ohledem na tento fakt jsem formulář ponechal jako součást aplikace a se vzdáleným serverem se pouze naváže spojení pro zaslání obsahu.

Kapitola 5

Realizace výsledné aplikace

Na řadu přichází implementační část, kde popisuji použité postupy z programátorského úhlu pohledu. Není mým záměrem vysvětlovat každý řádek kódu, ale postihnout hlavní části programu a zmínit některé zajímavé detaily. I s těmito prioritami se ale snažím o systematický přístup k dokumentaci celého procesu vývoje této aplikace, jíž jsem dal název Typing Stats. Programování jsem rozdělil na několik fází. Začal jsem od základních prvků a postupně na ně nabaloval další funkčnost. Implementaci jsem prováděl po těchto etapách:

- Tvorba jednoduchého doplňku bez vlastní funkčnosti (integrace ovládacích prvků do Firefoxu)
- Detekce psaného textu, jeho ukládání a zobrazení (tzv. „keylogger“)
- Zpracování textu a ukládání statistických dat
- Tvorba uživatelského rozhraní a přidávání nových funkcí aplikace

5.1 CoffeeScript místo JavaScriptu

Jak jsem již dříve uvedl, doplňky Firefoxu jsou klientsky orientovanou webovou aplikací, a proto jejich funkční část představují zdrojové kódy JavaScriptu. Existuje však poměrně nový programovací jazyk CoffeeScript, který je jakousi abstrakcí nad JavaScriptem a zjednodušuje zápis zdrojového kódu. Výsledek je pak přeložen do JavaScriptu. Proto jsem se pro implementaci rozhodl použít právě CoffeeScript.

Tato volba pro mě tedy neznamená získání nové funkčnosti, ale pohodlnější zápis syntaxe. Bloky lze oddělovat odsazením místo složených závorek, odpadá nutnost používání středníků a zjednodušená je i definice funkcí. Více lze nalézt v knižním průvodci tímto jazykem [2].

5.2 První krůčky k vlastnímu doplňku

Než jsem se pustil do sběru a vyhodnocování dat, bylo pro mě první výzvou udělat jednoduchý instalační balíček, který by přidal novou položku do menu Firefoxu a ikonku na nástrojovou lištu. Vytvořil jsem základní adresářovou strukturu a do souboru `chrome.manifest` uvedl příslušné cesty. V `install.rdf` jsem vyplnil informace o mém doplňku a protože se identifikátor zadává ve stejném formátu jako e-mailová adresa, zvolil jsem ID `typing-stats@lukasturek.org`.

V souboru `overlay.xul` jsem definoval novou položku v menu přidáním elementu `menuitem` do nadřazeného prvku `menupopup` a podobným způsobem jsem přidal také ikonu na lištu. Obrázek pro tuto ikonu je uložen ve složce `skin` a načítá se ze souboru `overlay.css`. Interakci obou vytvořených prvků s uživatelem jsem zajistil voláním funkce `openWindow()` po kliknutí na ně. Tímto jsem sice dosáhl pouze otevření prázdného okna, ale první doplněk modifikující prohlížeč byl na světě a mohl jsem začít s vytvářením jeho hlavní náplně.

5.3 Monitoring a ukládání dat

Následující část je popisem obsahu souboru `monitoring.coffee`, resp. `monitoring.js`. Tento skript je načítán ze souboru `overlay.xul`, takže je spouštěn spolu s prohlížečem, a jeho úlohou je sběr dat od uživatele.

5.3.1 Zachycení klávesy

K detekci stisku klávesy využívám vestavěnou funkci JavaScriptu `addEventListener()`, která na konkrétní události váže požadovanou akci. V tomto případě se jedná o událost `keypress`. V oblužné funkci se podle kódu znaku rozhodne, jak jej dále zpracovat.

Alfanumerické znaky se ze své kódové reprezentace převedou na řetězec. Z množiny řídicích znaků se sleduje pouze backspace, kde se z uloženého řetězce napsaného textu (pokud není prázdný) odebírá poslední znak a ukládá do proměnné. Když se u alfanumerického znaku zjistí, že předchozím byl backspace a nebyl stisknut více než pět krát (mazání větší části textu se nepovažuje za opravení překlepu), pak se vytváří záznam o chybě. Tyto záznamy jsou reprezentovány dvourozměrným polem.

Časovou značku úhozu získávám funkcí `getTime()`. Její návratovou hodnotou je čas v milisekundách od 1.1.1970, a tak lze interval mezi dvěma úhozy vypočítat pouze odečtením dvou takto obdržených hodnot.

5.3.2 Časovač

Prvek časovače byl zaveden kvůli potřebě hlídat uplynulou dobu od posledního stisku klávesy a počítá také celkový čas psaní. Spouštěn je spolu se zahájením nového záznamu při prvním stisku klávesy a zastaven po detekci kliknutí myší nebo vypršení časového intervalu pět sekund od posledního stisku klávesy.

Časovač jsem konstruoval zavedením funkce `setInterval()`, která zajistí pravidelné volání mnou definované funkce. V mém případě se tímto každou sekundu inkrementuje proměnná uchovávající informaci o počtu sekund od posledního stisku klávesy. Tato proměnná je globální a při každém stisku klávesy je vynulována. Pokud se porovnáním zjistí, že přesáhla nastavenou hodnotu, časovač je zastaven a přechází se k vyhodnocení textu a uložení statistických údajů. Z vnějšku může být přerušen zmíněným kliknutím myší, a proto je sledována událost `click` pomocí `addEventListener()`, podobně jako u stisku kláves.

5.3.3 Zápis do souboru

Ukládání dat jsem implementoval s využitím výhod modelu XPCOM a jeho rozhraní pro práci se soubory. Za cílovou oblast jsem zvolil profilovou složku Firefoxu, jejíž umístění lze automaticky zjistit metodou `get()`. V profilové složce pro přehlednost vytvořím vlastní složku `TypingStats` (pokud ještě neexistuje, tj. při prvním spuštění) a zde se budou shromažďovat všechny soubory vytvořené mou aplikací.

Soubory jsem se nejdřívě pokoušel vytvářet funkcí `initWithPath()`, kde jsem jako atribut zadával relativní cestu ve tvaru „složka/soubor“. Nemilým překvapením pak bylo, když jsem zjistil, že tento způsob funguje na operačním systému Linux, zatímco Windows si s tímto zápisem neporadil. Problém spočíval v odlišné reprezentaci cesty na těchto platformách, protože Windows používá zpětné lomítko a tento zápis nedekódoval požadovaným způsobem. Řešením bylo inicializovat cestu k souborům funkcí `append()` s následným dotazováním se na existenci souboru a případné vykonání `create()`.

Data jsou ukládána ve formátu XML a pro pohodlnou práci s nimi je zapotřebí parser. Existují i specializované knihovny, ale pro mé účely jsem zvolil `DOMParser`, který je ve Firefoxu vestavěný. Pro každý nový záznam o psaní se vytvoří kořenový prvek obalující elementy, které reprezentují sledované veličiny a v těchto elementech se dále vytvoří textové uzly s hodnotami těchto veličin. Nakonec přichází ke slovu serializér, který tuto abstraktní reprezentaci převádí na řetězec pro zápis.

Zápis dat zde původně měla obstarat funkce `fwrite()`, což je jedna ze standardních funkcí JavaScriptu pro práci se soubory. Všiml jsem si ale toho, že se občas data zapisují jen částečně, a to hlavně kvůli porušeným tagům vedlo na nevalidní XML soubory. Proto jsem začal pátrat po asynchronní variantě zápisu a našel jsem modul `NetUtil`, jímž jsem původní funkci nahradil.

5.4 Generování statistik

Data jsou vyhodnocena a graficky zpracována souborem `generate.coffee`, resp. `generate.js`. V následujícím textu přiblížím, jak byly implementovány jednotlivé části při realizaci uživatelského rozhraní a jeho součástí.

5.4.1 Zajištění multijazyčnosti

Primárně jsem vyvíjel tento projekt v českém jazyce a dodatečně byl přeložen do angličtiny, aby se mohl rozšířit i mezi zahraniční uživatele. Přepínání mezi oběma jazyky nabízím v sekci nastavení. Po změně jazyka dojde k přesměrování do adresáře pro příslušnou jazykovou verzi. Abych od tohoto principu uživatele odstínil, přecházím ihned po načtení zpět do sekce nastavení, což je realizováno proměnnou v URL adrese, která je po inicializaci JavaScriptem parsována. Zvolený jazyk bude zachován i pro příští spuštění díky komponentě XPCOM `preferences-service` pro tvorbu a editaci uživatelských nastavení ve Firefoxu.

Prioritní je vždy nastavení uživatele, musel jsem ovšem rozhodnout, jakým způsobem se provede výběr výchozího jazyka po prvním spuštění. To jsem nakonec implementoval tak, že se z globální proměnné `navigator.language` zjistí nastavení jazykového prostředí prohlížeče a regulárním výrazem se porovná na shodu prvních dvou znaků s písmeny „cs“, což odpovídá češtině. Pokud nedojde ke shodě při porovnání, bude výchozím jazykem angličtina. Protože jsou udržovány HTML zdrojové kódy zvlášť pro každý jazyk, načtení konkrétní jazykové verze představuje spuštění souboru z příslušné složky.

Velká část obsahu je však generována dynamicky JavaScriptem, a v tomto případě jsem zvlášť oddělil funkční kód od textu. Ve složkách s HTML soubory jsem u obou jazykových verzí pro tento účel vytvořil soubor `lang.js`, který je načítán relativní cestou a obsahem jsou proměnné s potřebnými textovými definicemi. Na tyto proměnné se odkazuji ze souboru `generate.js` a právě díky relativní cestě je zaručen výběr momentálně aktivní jazykové verze.

5.4.2 Dynamické načítání obsahu

Po spuštění generátoru statistik se do inicializovaného okna načítá soubor `TypingStats.html` s obsahem kostry uživatelského rozhraní. Většina HTML kódu je poté dynamicky generována pomocí JavaScriptu. Tímto způsobem jednak dosahuji toho, aby celé prostředí bylo více interaktivní, např. zbarvením tlačítka submenu po kliknutí, ale v některých částech to bylo přímo nevyhnutelné, jako třeba u vytváření tabulky s opravovanými chybami. Pro tyto účely používám práci s DOM modelem, kde jsem jednotlivým prvkům přiřadil identifikátor a pomocí něj je pak vyhledávám metodou `getElementById()`.

Takto získaným objektům jsem chtěl nastavovat atribut `innerHTML` pro naplnění obsahem. Při automatizovaném procesu kontroly během nahrávání doplňku jsem však byl upozorněn varováním, že používání `innerHTML` v kombinaci s proměnnými může znamenat snížení výkonnosti a bezpečnostní rizika. Proto jsem v místech, kde nebylo nutné vkládat HTML tagy, nahradil tento atribut za `textContent`, kde nedochází k překladu na HTML kód a případné tagy jsou vypsány jako čistý text.

5.4.3 Udržování kontextu

V době návrhu této aplikace jsem ještě nevěděl o výhodách uživatelských preferencí uložitelných přímo ve Firefoxu. Prototyp Typing Stats z tohoto důvodu nejdříve ukládal nastavení uživatelského rozhraní do vlastního konfiguračního souboru. Ideou bylo udržovat v souboru `config.xml` zvolené parametry pro načítání grafu, které by tím byly uchovány do dalšího spuštění.

Tento koncept jsem vyměnil za čtění a zápis preferencí Firefoxu, což je daleko efektivnější. Jediným voláním funkce lze vytvořit (resp. přepsat) předvolbu a po ověření její existence funkcí `prefHasUserValue()` z ní lze následně číst. Výhodou je, že se nemusí při každé změně ukládat znovu všechny preference, jak tomu bylo v případě konfiguračního souboru, ale pouze konkrétní požadovaná proměnná.

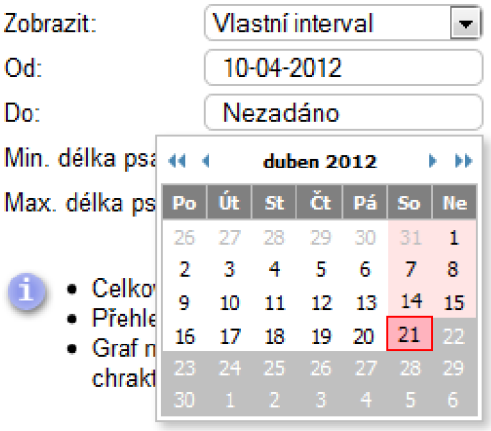
5.4.4 Kalendář pro výběr data

Kalendář se objeví při volbě vlastního časového intervalu omezujícího data pro zobrazený graf. Jeho funkci zajišťuje volně dostupná JavaScriptová knihovna Tigra Calendar, kterou jsem si pro své účely upravil.

Mou první úpravou je způsob inicializace kalendáře, protože v původní podobě byl určen pro spouštění z textového okna, do kterého se datum po jeho výběru zapisovalo ve formátu „měsíc/den/rok“. Já jsem nahradil textové okno tlačítkem a ve formátu jsem přehodil měsíc a den, což je obvyklejší zápis pro Evropu.

Větším zásahem do knihovny bylo odlišení dnů z budoucnosti, aby nebyla možná jejich volba a tím pádem zbytečné generování prázdných statistik. To jsem vyřešil přidáním kódu JavaScriptu, který u zobrazovaného dne určí, zda jeho datum není pozdější než dnešní. Tyto dny se pak nezobrazí jako odkazy a dále jsou pomocí CSS barevně odlišeny.

Nakonec jsem vedle anglické verze kalendáře vytvořil i český překlad, mírně upravil soubor s kaskádovými styly pro větší barevnou sladěnost s mým rozhraním a knihovna byla připravená k použití. Zvažoval jsem i omezení výběru z historie na nejstarší datum, o němž je vytvořen záznam statistik, ale díky přepočítávání rozlišení grafu je toto omezení zbytečné, což se potvrdilo při testování, kdy jsem neměl problém vykreslit grafy i v časových intervalech větších než sto let.



Obrázek 5.1: Kalendář pro výběr data

5.4.5 Příprava grafu a jeho vykreslení

Načítání grafu jsem implementoval ve funkci `showChart()` volané při inicializaci a dále při každé změně parametrů grafu. Na počátku se zkontroluje, jestli nebyl zvolen záporný interval (konečné datum starší než počáteční) a případně se tato chyba ošetří oznámením uživateli. Po této kontrole se pokračuje počítáním potřebných údajů a jejich předáváním přes rozhraní knihovny `Bluff` pro vykreslení grafu. U histogramu je pro sloupcový graf vytvořena nová instance `Bluff.Bar`, pro ostatní typy grafu `Bluff.Line`. Graf je vázán ke konkrétnímu HTML objektu `canvas`, do kterého se vykresluje. Proces přípravy a vykreslení grafu zahrnuje tyto úkony:

- Určení celkového počtu bodů
- Přiřazení příslušných hodnot jednotlivým bodům
- Výpočet frekvence popisků časové osy na základě hustoty bodů
- Generování popisků
- Předání hodnot knihovně Bluff pro vykreslení

Co se týče určení počtu bodů a přiřazení hodnot, je u histogramu situace jednodušší, neboť se jeho x-ová osa s výběrem časového období dynamicky nemění. U ostatních typů grafu nejdříve počítám ve vlastní funkci `CalcDifference()` rozdíl koncového a počátečního data v jednotce dní. U příliš malého nebo naopak velkého rozlišení (viz. 4.7.2 v části návrhu) je místo dní zvolena nová jednotka a množství bodů znovu přepočítáno. Pak je inicializováno pole o délce rovné tomuto počtu. Pokračuje se postupným procházením všech záznamů o psaní, z nichž se zpracují jen ty odpovídající právě zvolenému intervalu, a podle typu grafu se vybere sledovaná veličina. Její hodnota se podle stáří záznamu přičte k příslušnému bodu grafu, jehož index počítám stejnou funkcí `CalcDifference()`. Pro grafy rychlosti a chybovosti, kde se mají zobrazit průměrné hodnoty, se navíc uchovává pole s informací o počtu záznamů jednotlivých bodů, aby se následně tímto číslem mohly vydělit získané součty.

Ve funkci `setLabels()` vytvářím popisky grafu na základě počtu bodů a zobrazované jednotky. Pro jednotku hodin je to dvouciferné číslo reprezentující jejich počet, pro dny je to datum ve formátu „dd.mm.“, u měsíců jejich třípísmenná zkratka a poslední možností je rozlišení číslem roku. Frekvenci těchto popisků počítám na základě hustoty bodů tak, aby se vzájemně nepřekrývaly. Když jsou data i popisky pro graf nachystána, rozhodne se podle typu grafu už jen o jeho nadpisu a barvě a voláním funkce `draw()` se vykreslí.

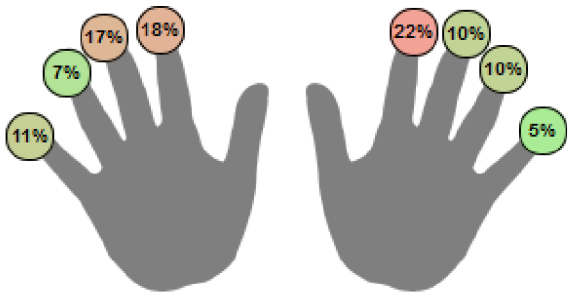
5.4.6 Výpis opravovaných znaků

V sekci `Chyby` se zobrazují nejčastěji opravované znaky. Tato statistika je vedena za celou dobu používání programu, zpracovávají se proto všechny záznamy o psaní. Aby se při přechodu mezi sekcemi grafů a chyb nemusely soubory znovu načítat, nedochází k přesměrování na novou stránku, ale pouze se JavaScriptem přepíše obsah hlavního okna funkcí `loadMistakes()`.

Během procházení záznamů o chybách vytvářím pole trojic „původní znak, nahazený znak, počet výskytů“, ve kterém se nejdříve hledá výskyt existující dvojice „původní znak, nahrazený znak“ a v případě nalezení se inkrementuje počet výskytů. V opačném případě se vytvoří nový záznam s četností rovnou jedné. Takto vzniklé pole seřadím podle počtu výskytů vestavěnou funkcí `sort()`, jíž jako parametr předávám vlastní funkci `sortNumber()` definující způsob řazení prvků. Nakonec se v cyklu pro každý prvek pole vytvoří nový řádek tabulky a díky nastaveným identifikátorům jednotlivých buněk pak v souboru CSS definuji odlišné barvy pro zobrazení původních a nahrazených znaků.

5.4.7 Určení chybovosti prstů a její prezentace

Vedle opravovaných znaků se ve stejné sekci zobrazuje i podíl prstů na chybách. Na siluetě rukou je u jednotlivých prstů v kolečku tato hodnota uvedena v procentech a barva pozadí se podle ní mění od červené po zelenou. Výjimkou jsou palce, protože ovládají pouze mezerník a navíc nelze určit, kterým palcem byla mezera stisknuta, a proto se nevyhodnocují. Náhledem na příslušnou část uživatelského rozhraní je obrázek 5.2.



Obrázek 5.2: Rozložení celkového počtu chyb na jednotlivé prsty

Výpočet chyb každého prstu se provádí při procházení všech záznamů spolu s hledáním často opravovaných znaků. Vybírá se vždy nahrazený znak a hledá se jeho příslušnost k prstu, který jej píše. Zde záleží na rozložení klávesnice, které nelze detekovat přímo, ale v závilosti na zvoleném nastavení v uživatelském rozhraní se vybere, podle jakého rozložení má být chybovost prstů hodnocena. Po zjištění hodnoty této proměnné uložené jako

uživatelská preference Firefoxu `extensions.typingstats.keyboardLayout` přiřazuji znak k prstu vlastní funkcí `QWERTYchars()` nebo `QWERTZchars()`. Obě funkce vrací číslo prstu při shodě s některou možností ve výčtu znaků v konstrukci `switch`. Nakonec se pro každý index pole přepočítá chybovost na podíl ve vztahu k celkovému počtu chyb.

Kruhy, v nichž se chybovost zobrazuje, jsou realizovány jako HTLM prvky `DIV` a barva pozadí je nastavována funkcí `rgb(150+R, 255-R, 150)`, kde R nabývá hodnot 0-105 a toto rozmezí je přepočítáno na rozsah chybovosti 0-25%, aby pro 25% a víc byla výsledná barva červená. Základní hodnotu 150 jsem zvolil kvůli zesvětlení pozadí a lepší čitelnosti vnitřního textu.

5.5 Odeslání dat z formuláře

Jak jsem zmínil u návrhu aplikace, je načítání vzdáleného obsahu do doplňku bezpečnostním rizikem, a proto jsem formulář implementoval lokálně s odesíláním dat metodou `POST`. Vytvořil jsem objekt `XMLHttpRequest` a sledováním událostí `onload` (při úspěšném spojení) a `onerror` (při neúspěchu) mohu reagovat na odeslání či neodeslání dat z formuláře příslušnou zprávou pro uživatele. Abych obsah nemusel vybírat ručně, dělám to vestavěnou funkcí `FormData()`, která jej zrovna převede na potřebný formát pro vstup metody `POST`.

Serverovou část jsem umístil na mé školní webové stránky. Jedná se o soubor `form.php`, který přijímá odeslaná data a zpracovává jednotlivé proměnné (předmět, kontaktní e-mail, text zprávy). Tento obsah si následně nechávám posílat na svou e-mailovou adresu PHP funkcí `mail()` a zároveň jej zálohuji vytvořením záznamu v MySQL databázi.

5.6 Úklid při odebírání doplňku

Smazání dat je možno provést kdykoli v uživatelském rozhraní. Aby však soubory na disku zbytečně nezabíraly místo po odinstalování doplňku, ptám se uživatele, zda si je přeje odstranit. Automaticky se toto neděje, aby se při eventuální opětovné instalaci zobrazily i původní statistiky.

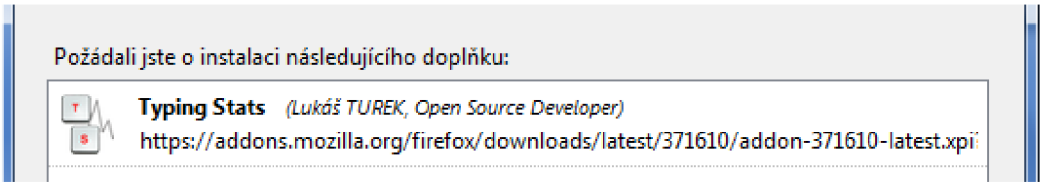
Správce doplňků Firefoxu poskytuje API s funkcemi pro práci s doplňky a mezi nimi také funkci `addAddonListener()`, díky které mohu reagovat na odinstalování doplňku. Je však potřeba odlišit ID doplňku, což jsem nejdříve opomněl a výsledným nepříjemným efektem bylo, že byl na smazání statistických dat uživatel dotazován při odebírání jakéhokoli doplňku. Dialog pro potvrzení jsem realizoval jako `confirm()` a při kladné volbě je smazána celá složka `TypingStats`.

5.7 Podepsání certifikátem a publikování

První verzi mého doplňku jsem šířil pouze prostřednictvím osobního webu, abych získal reakce od svého okolí, a pak jsem se rozhodl publikovat jej oficiálně na webu Mozilly.

Protože se standardně při instalaci doplňku zobrazuje v dialogovém okně text „autor neznámý“, hledal jsem způsob, jak na tomto místě uvést své jméno. K tomu je potřeba mít vlastní certifikát, kterým se zdrojový kód podepíše. Většina z nich je placených, ale existuje polská společnost Certum, která pro open source vývojáře poskytuje certifikáty zdarma. Této možnosti jsem využil a zaregistroval jsem se pomocí internetového formuláře. Po ověření mých osobních údajů mi byl zaslán certifikát autentizující mou identitu. Pomocí nástrojů `certutil` a `signtool` jsem takto získaným certifikátem podepsal výsledný XPI

soubor. Pro ověření mého podpisu je potřeba mít v seznamu certifikátů ve Firefoxu mezi autoritami přidán `Certum Level III CA`, který je k dispozici ke stažení mezi kořenovými certifikáty společnosti Certum [1].



Obrázek 5.3: Instalace s úspěšným ověřením mého podpisu

Automatizovaná kontrola mého doplňku při nahrávání na stránky Mozilly skončila s několika varováními, proto jsem musel udělat ještě ve zdrojovém kódu několik změn. Nejčastějším problémem bylo používání atributu `innerHTML` v kombinaci s proměnnými, což Mozilla nedoporučuje kvůli výkonu a bezpečnosti. Dále jsem byl upozorněn na dynamické nastavování atributu `onclick` funkcí `setAttribute()`, které jsem nahradil přípustnou variantou volání funkce `onclick()` a další upozornění se týkalo přepisování globálních proměnných. Všechny tyto aspekty jsem postupně vyřešil, aby se můj doplněk nahrál bez chyb a varování.

Kapitola 6

Vyhodnocení zpětné vazby od uživatelů

Po dokončení implementace jsem rozšířil můj doplněk mezi první uživatele a začal jsem s vyhodnocováním mé práce. Toho jsem se ujal ze dvou různých hledisek. Za prvé jsem požádal uživatele, aby mi poskytli svá statistická data vygenerovaná aplikací. Získané vzorky jsem následně srovnával, abych z nich vyvodil patřičné závěry o vyskytujících se trendech ve vztahu k uživatelům s různou úrovní psaní na klávesnici. Za druhé mě zajímaly názory na výsledný produkt a návrhy na zlepšení, které by se pro mě staly podnětem k budoucímu vývoji a vylepšování stávající podoby tohoto doplňku pro Firefox.

Čekání na kontrolu před publikováním na stránkách Mozilly bohužel celkově trvalo více než tři týdny, protože nahráním nové verze jsem se opět zařadil na konec fronty čekajících doplňků ke schválení. Během této doby však stažení bylo možné z neveřejného odkazu na doplněk (s nepřehlédnutelným varováním o riziku poškození počítače), a tak jsem zatím aplikaci šířil alespoň v mém okolí prostřednictvím tohoto odkazu. V době psaní tohoto textu uplynulo 7 dní od zveřejnění mezi oficiálními doplňky a v přehledu statistik má aktuálně můj doplněk 223 stažení. Po dlouhém čekání na kontrolu první verze mě pak příjemně překvapila rychlá reakce na další nahranou verzi, která byla publikována již za dva dny.

6.1 Porovnání dat a zjištěná fakta

Získat dostatečné množství referenčních dat nebylo jednoduché, neboť bylo podmíněno hned několika faktory – aby potenciální účastník tohoto experimentu používal internetový prohlížeč Firefox, chtěl si doplněk nainstalovat a byl ochotný zaslat mi zpět získaná data. Mezi mnou oslovenými bylo nakonec i hodně lidí, kteří sice byli ochotní se zapojit, ale protože upřednostňovali jiné internetové prohlížeče, získal jsem od nich jen minimální množství dat. Nakonec jsem vybral reprezentativní vzorek dvanácti uživatelů, z nichž tři píší hmatovou technikou, všichni jsou Češi a píší s rozložením kláves typu QWERTZ a jejich data byla získána za období delší než týden. Samozřejmě první naměřená data jsem získal vlastním používáním během vývoje a testování aplikace. Hmatovou techniku psaní na klávesnici používám již 8 let, a proto jsem chtěl tento nástroj nejdříve otestovat sám na sobě před jeho uvolněním do světa.

Zajímavé bylo sledovat vývoj chybovosti prstů v průběhu používání programu. Všiml jsem si totiž, že tyto hodnoty vždy konvergují určitým směrem. To mě přivedlo k myšlence, že vlastně nelze jednoznačně říci, že by uživateli dělal výrazně větší problém nějaký prst

ve srovnání s ostatními. Jan Králík v roce 1983 publikoval svůj výzkum o četnosti písmen v českém textu [5]. Analyzoval text s více než třemi miliony znaků a z něj vypočítal poměrné zastoupení písmen. Výsledky tohoto výzkumu jsem použil a podle příslušnosti jednotlivých znakům k prstu, který tento znak píše (podle českého standardu rozložení kláves QWERTZ), jsem pro každý prst vypočítal jeho podíl na aktivitě při psaní českého textu. Výsledky prezentuji v tabulce 6.1. Pozoroval jsem, že čím více dat od uživatele bylo získáno, tím více se podíl prstů na chybách blížil tomuto vypočítanému podílu na aktivitě při psaní. U 719 celkově opravených chyb byla maximální odchylka 3%.

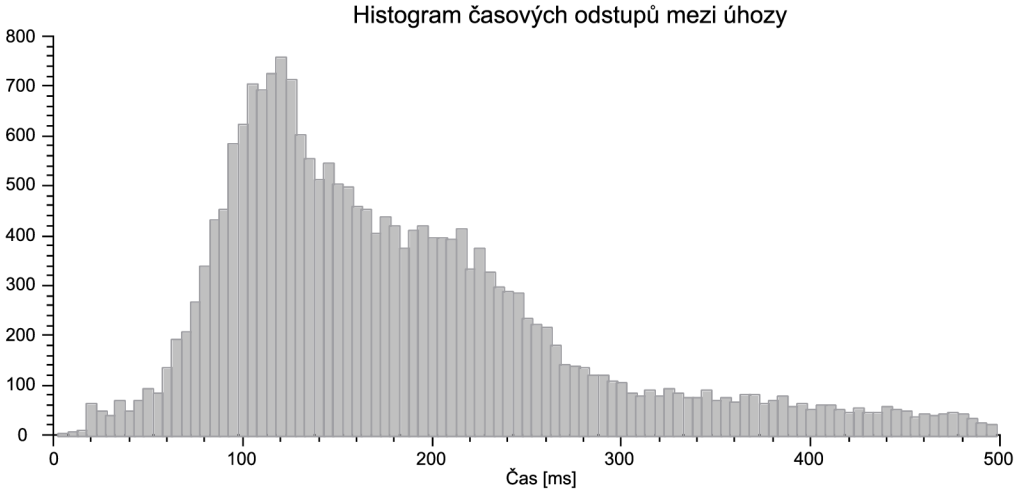
Prst	**Podíl na psaném textu**
Levý malíček	9,7752 %
Levý prsteníček	5,4055 %
Levý prostředníček	14,4604 %
Levý ukazováček	18,4441 %
Pravý ukazováček	22,4701 %
Pravý prostředníček	11,3594 %
Pravý prsteníček	13,8747 %
Pravý malíček	4,2106 %

Tabulka 6.1: Podíl prstů na celkové aktivitě při psaní českého textu

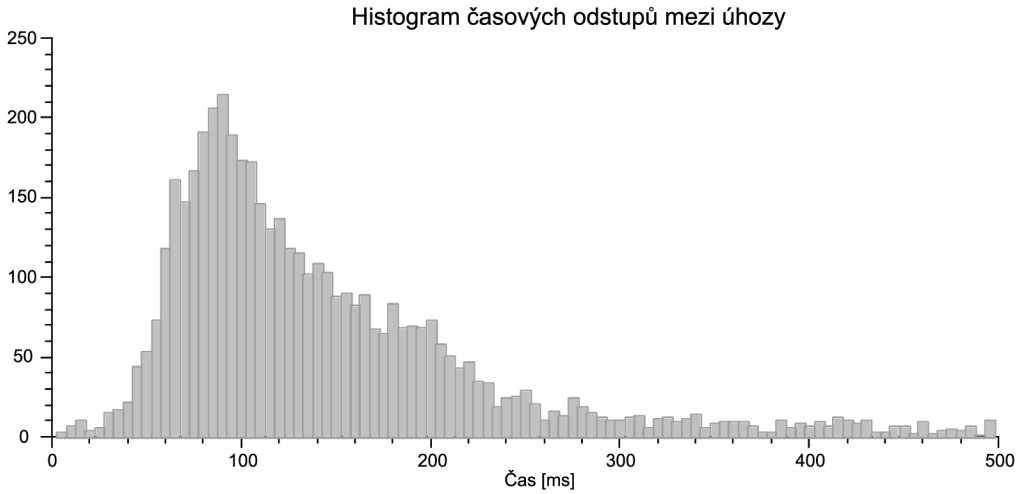
To by nebylo nic překvapivého u uživatelů, kteří nepíší hmatovou technikou, ale pouze několika prsty se sledováním klávesnice. Zde je totiž chybovost prstů počítána uměle a nemá vypovídající hodnotu. Stejný vývoj však byl sledován i u uživatelů, kteří píší hmatovou technikou již více let. Toto zjištění ve svém důsledku znamená, že u uživatelů neovládajících hmatovou techniku psaní stejně jako u uživatelů píšících touto technikou není možné říci, že by jim dělal větší problémy nějaký konkrétní prst. Překlepy jsou dělány rovnoměrně všemi prsty a na základě momentální koncentrace uživatele se pouze mění jejich intenzita. Mou osobní domněnkou, kterou jsem zatím neměl možnost prokázat, je to, že naměřené hodnoty by se více lišily u jedinců, kteří se právě učí psát všemi deseti prsty a některé klávesy ještě nemají příliš zažité, pletou si konkrétní klávesy s jinými apod. Ve vzorku uživatelů, na kterých jsem mé testování prováděl, však bohužel nikdo takový nebyl.

Naopak velkou vypovídající hodnotu mělo srovnání histogramu časových odstupů mezi úhozy u uživatelů s různou úrovní psaní. Je dobře patrné, že lidé ovládající hmatovou techniku psaní mají celkový vzhled histogramu užší, protože píší plynuleji. Naproti tomu u lidí píšících se sledováním klávesnice je tento histogram vizuálně širší vlivem občasného, byť krátkého, hledání klávesy. Tento trend lze pozorovat porovnáním grafů z obrázku 6.1 a 6.3, které byly získány z dat dvou různých uživatelů za období dvou týdnů. První obrázek ukazuje histogram s větším rozptylem hodnot při psaní šesti prsty se sledováním klávesnice a na druhém je vidět menší rozptyl při psaní hmatovou metodou.

Analýza získaných dat také ukázala, že zahrnutím velmi krátkých textů není zkreslení tak velké, jak jsem předpokládal. Při návrhu jsem vycházel z toho, že při psaní delšího textu se výkonnost a koncentrace postupně snižuje. Po odfiltrování textů kratších než 10 sekund se rychlost psaní v průměru snížila jen o 2%. Větší rozdíly byly vidět při posouvání této spodní hranice, kdy někteří uživatelé u delších textů měli dokonce vyšší rychlost. Tuto bilanci připisuji faktu, že můj původní předpoklad odpovídal spíše opisovanému textu, zatímco u vlastního textu od uživatele dochází k častějším přestávkám i u krátkých textů.



Obrázek 6.1: Uživatel používající při psaní 6 prstů se sledováním klávesnice



Obrázek 6.2: Uživatel píšící hmatovou technikou

6.2 Rychlost psaní vymýšleného vs. opisovaného textu

Jak jsem zmínil výše, je patrný rozdíl mezi textem, který uživatel píše během jeho vymýšlení, a textem opisovaným. Poprvé jsem se této problematiky dotknul u zkoumání vlivu délky textu na průměrnou rychlost. Zjišťoval jsem, jaké jsou rozdíly mezi hodnotami naměřenými mou aplikací při běžném používání internetového prohlížeče a hodnotami při opisování textu, kdy se pisatel soustředí pouze na co nejrychlejší reprodukci textu.

Provedl jsem krátký výzkum, kdy jsem uživatele mé aplikace nechal minutu opisovat pro ně neznámý text, vyhodnotil čisté úhozy za minutu (bez penalizace chyb odečítáním více úhozů) a srovnal výsledky s jejich průměrnou rychlostí psaní ve Firefoxu. Ukázalo se, že lidé píšící hmatovou technikou psaní mají při opisu výrazný nárůst rychlosti proti statistickým datům z mého doplňku, a to až o 300%. Naopak účastníci experimentu sledující při psaní klávesnici měli výkony srovnatelné či spíše nižší, s výchylkami v desítkách procent. Mnou

naměřené hodnoty ukazují, že při opisu textu hmatovou technikou se proti vymýšlenému textu rychlost dramaticky zvyšuje. U lidí bez zažitých návyků na psaní všemi deseti prsty je výhoda předem zadaného textu vykrácena nutností sledovat zároveň předlohu, klávesnici a monitor, přičemž se často zdrží i tím, že se ztratí v opisovaném textu.

Z výsledků tohoto výzkumu plyne, že rychlost psaní generovaná mým doplňkem je spíše orientační, protože zahrnuje schopnost rychlé tvorby myšlenek a jejich následné převedení na text. Přitom ale nevypovídá o skutečném potenciálu uživatele a jeho maximální rychlosti psaní, kterou je schopen vyvinout při opisování textu z předlohy. V případě uživatelů sledujících při psaní klávesnici se výsledky více blíží reálným schopnostem, ale to je dáno nahrazením prodlev při vymýšlení textu jiným typem prodlev vznikajících při opisování.

6.3 Reakce a návrhy na zlepšení

Bezprostřední dojmy uživatelů byly vesměs pozitivní. Hodnocení se mi dostávalo osobně nebo po různých komunikačních kanálech a formulář zpětné vazby proto zatím nebyl příliš využit. Konkrétních návrhů na změnu či zlepšení jsem se zatím nedočkal, ale díky těmto prvním uživatelům jsem alespoň odhalil několik drobných chyb. Např. formulář zpětné vazby místo potvrzovací zprávy vypisoval název proměnné, která tuto zprávu obsahovala. Od cizích uživatelů zatím nemám žádné zprávy, což přisuzuji zejména čerstvému oficiálnímu zveřejnění, čímž pádem tito uživatelé ještě nemají dostatečné množství statistických dat k hodnocení, zda byl pro ně můj doplněk přínosem či nikoli. Za pozitivní zpětnou vazbu ale považuji i postupný nárůst počtu stažení a denních uživatelů.

Když píši o reakcích, potěšil mě také zájem týmu Softpedia, jež na svůj server umisťuje freeware produkty ke stažení. Od těchto lidí mi přišla zpráva o zařazení mého doplňku do jejich databáze obsahující více než milion aplikací. Po ověření jejich editorem mi byl zaslán certifikát stvrzující, že je obsah programu neškodný, bez výskytu virů, spyware a obtěžujících reklam.

Obrázek 6.3: Certifikace serverem Softpedia

Podněty pro budoucí vývoj jsem tedy nakonec získal spíše porovnáním získaných dat od uživatelů, kteří byli ochotní se podrobit mým výše popsaným experimentům. Zejména zvažuji, zda ponechat vyhodnocování chybovosti prstů. K definitivnímu rozhodnutí však budu potřebovat i data od uživatelů začínajících nebo méně pokročilých v psaní hmatovou technikou. Mohlo by se totiž ukázat, že v jejich případě tato veličina více vybočuje od vypočítané aktivity prstů při psaní. V opačném případě bych se pokusil nahradit sledování chybovosti prstů za jejich rychlost a na základě rozdílů mezi uživateli opět zkoumat efektivitu. Pro budoucí verzi mě také napadlo přidat sledování rychlosti stisku jednotlivých znaků, které jsou nyní souhrnně vyhodnocovány v histogramu.

Kapitola 7

Závěr

V této práci jsem popsal tvorbu nástroje pro analýzu psaní v podobě doplňku internetového prohlížeče Firefox. Potřebnými prerekvizitami pro splnění tohoto úkolu bylo studium hmatové techniky psaní a prostoupení do tajemství doplňků Firefoxu. Psaní na klávesnici je pro mě blízkou disciplínou, protože jsem měl na základní škole možnost se zúčastnit několika soutěží, a tak jsem jsem se snažil vcítit do role lektora a zaměřil se na atributy, které jsou při psaní důležité. Tvorba doplňků pro Firefox pro mě naopak byla naprosto neznámou doménou, a tak jsem začínal od nejjednodušších návodů a příkladů.

Na základě nabytých znalostí jsem se pustil do návrhu a implementace nového nástroje pro analýzu psaní. Výsledkem je doplněk Firefoxu s názvem Typing Stats, který podle detekovaného vstupu z klávesnice generuje grafy sledující rychlost, chybovost, plynulost a celkový čas věnovaný psaní. Dále jsem se zaměřil na často opravované chyby a jejich grafickou prezentaci pro uživatele. K ověření mé identitiy jsem si u společnosti Certum nechal vytvořit certifikát pro podepisování open source zdrojových kódů a doplněk jsem publikoval na oficiálních stránkách Mozilly, kde má po týdnu 223 stažení.

V době čekání na prověření editorem jsem prováděl sběr dat od uživatelů z mého okolí a zkoumal tyto vzorky ve vztahu k jejich stylu psaní, který uvedli. Zajímavým zjištěním bylo, že chybovost prstů konverguje k četnosti jejich používání při psaní českého textu (všichni účastníci experimentu byli Češi), a to jak u uživatelů píšících již delší dobu hmatovou technikou, tak u osob sledujících při psaní klávesnici. Bohužel se mi nepodařilo najít nikoho, kdo by se psaní všemi deseti prsty právě učil a potvrdil by tak mou domněnku, že se u takového člověka rozložení chybovosti prstů bude více lišit, protože ještě v paměti nemá dobře zafixovanou polohu některých kláves. Získaná data však ověřila i některé předpokládané fakty, jako například rozdíl rychlosti u opisovaného textu od právě vymýšleného či užší histogram časových odstupů mezi úhozy u uživatele s hmatovou technikou psaní ve srovnání s uživatelem píšícím jen několika prsty.

Tento projekt bych chtěl dále udržovat a vylepšovat na podněty uživatelů. Možná by bylo vhodnější se místo chybovosti více zaměřit na plynulost psaní a pokud by se chybovost prokázala jako úměrná intenzitě používání prstů také v průběhu výuky psaní, kompletně bych tuto sekci nahradil.

Literatura

[1] General Certification Authority CERTUM [online]. http://www.certum.eu, 2012-05-02 [cit. 2012-05-02].

[2] Burnham, T.: *CoffeeScript: Accelerated JavaScript Development*. The Pragmatic Bookshelf, 2011, iSBN 1-934356-78-4.

[3] Dobson, A.: *Touch Typing in 10 Hours*. How To Books Ltd, 2006, iSBN 1857038274.

[4] Edwards, J.: *Build your own Firefox extension*. SitePoint Pty. Ltd., 2009.

[5] Králík, J.: Statistika českých grafémů s využitím moderní výpočetní techniky. *Slovo a slovesnost*, ročník 44, č. 4, 1983: s. 295–304, iSSN 0037-7031.

[6] Mozilla: Proces kontroly [online]. https://addons.mozilla.org/cs/developers/docs/policies/reviews, 2011-01-13 [cit. 2012-04-28].

[7] Mozilla: Security best practices in extensions - MDN [online]. https://developer.mozilla.org/en/Security_best_practices_in_extensions, 2011-02-04 [cit. 2012-04-25].

[8] Mozilla: Extension Versioning, Update and Compatibility - MDN [online]. https://developer.mozilla.org/en/Extension_Versioning,_Update_and_Compatibility, 2011-05-04 [cit. 2012-04-22].

[9] Nadběla, J.: *Deseti prsty na klávesnici*. Computer Media, 2006, iSBN 80-86686-66-3.

[10] Neugebauer, T.: *Profesionálem v administrativě*. Rubico, 2004, iSBN 80-7346-016-5.

[11] Reyes, M.: *Hacking Firefox*. Wiley Publishing, 2005, iSBN 0764596500.

[12] Zaviačič, J.: Škola ZAV [online]. http://www.zav.cz, 2012-04-05 [cit. 2012-04-20].