Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Návrh softwaru pro emulaci vykreslení webové stránky

DIPLOMOVÁ PRÁCE

Student : Miroslav Čech

Vedoucí : Ing. Jarmila Pavlíčková, Ph.D.

Oponent : Ing. Václav Pokluda

2017

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 20. dubna 2017 . . . . . .

Miroslav Čech

Poděkování

Na tomto místě bych rád poděkoval vedoucí mé práce za cenné rady a připomínky, a hlavně za to, že si na mne vždy našla čas, když bylo potřeba. Dále chci poděkovat svým kolegům. Nejen že mi umožnili věnovat se tvorbě této práce a že mi poskytli potřebnou infrastrukturu k jejímu dokončení, ale především za to, že mi navrhli takto zajímavé téma. V neposlední řadě mé díky patří vývojářům produktů, které jsem v této práci použil. Bez nich a jejich vůle zveřejnit své zdrojové kódy by se tato práce ubírala úplně jiným směrem.

Abstrakt

Autor této práce si klade za cíl navrhnout a vytvořit prototyp softwaru, který ukáže, jak se na daném zařízení zobrazí konkrétní webová aplikace. Nejprve je čtenáři popsán problém různorodosti zařízení přistupujících na web a úrovně podpory moderních webových technologií. Poté je přednesen způsob, jak vývojářům usnadnit testování napříč prohlížeči v různém hardwaru. V analytické části jsou popsány způsoby, jakými autor chce dosáhnout vytyčeného cíle, a také technologie, které je možné pro řešení použít. V předposlední části je uveden postup, kterým autor prototyp softwaru implementoval. Software je posléze podroben testům, které ukazují, že prototyp splňuje uvedené požadavky.

Klíčová slova

Webové technologie, webový prohlížeč, CSS, Javascript, Chromium, WebKit, Blink, mobilní zařízení

Abstract

The author of this thesis aims to analyze and develop a prototype of a software responsible for showing how the specific web application would be displayed by the concrete device. First, the problem of diversity of devices for accessing the web and of the level of support of modern web technologies is explained to the reader. Then the thesis offers a way how to help developers with testing across web browsers on different hardware. The analytical section describes the methods the author would like to use for accomplishing the goal, and also the set of suitable technologies. The penultimate section shows the manner the author implemented the prototype of the software. Software is then subjected to tests showing that the prototype meets the requirements.

Keywords

Web technologies, web browser, CSS, Javascript, Chromium, WebKit, Blink, mobile devices

Obsah

Obsah 1

Seznam obrázků 4

Seznam tabulek 5

1 Úvod 1

1.1 Cíle práce 2

1.1.1 Hlavní cíle 2

1.1.2 Vedlejší cíle 2

1.2 Cílová skupina 2

1.3 Použité metody 3

1.4 Struktura práce 3

2 Komentovaná rešerše informačních zdrojů 5

3 Popis současného stavu 8

3.1 Pochopení problému 8

3.1.1 Určení konkrétního zařízení uživatele 8

3.1.2 Zjištění schopností daného zařízení 8

3.1.3 Určení cílové množiny 9

3.1.4 Vizuální zpětná vazba 9

3.1.5 Rozdíl oproti referenčnímu zobrazení 10

3.1.6 Dodání statistických podkladů 10

3.2 Analýza existujících řešení 10

3.2.1 Broo 10

Plány na rozvoj 12

3.2.2 WURFL 12

3.2.3 Device Atlas 13

3.2.4 Can I use 13

3.2.5 Caniuse-api 13

3.2.6 Ostatní webové stránky s přehledem podpory technologií 14

3.2.7 Webový prohlížeč 14

3.2.8 WebPageTest 14

3.3 Obhajoba vlastního řešení 15

4 Analýza vývoje softwaru 17

4.1 Vstupní předpoklady práce 17

4.1.1 Technické předpoklady 17

4.1.2 Požadavky na zdrojový kód 18

4.2 Analýza uživatelských požadavků 19
4.2.1 Současné působení firmy 19
4.2.2 Cíle firmy 19
4.2.3 Funkční požadavky 20
4.2.4 Případy užití 20
4.2.5 Uživatelské scénáře 21
Získání vizuální podoby stránky 21
Získání DOMu webové stránky 22
Získání chyb funkčnosti 22
4.2.6 Nefunkční požadavky 22
4.3 Plán testů 23
4.3.1 Testovací prostředí 23
4.3.2 Testovací nápady 23
4.3.3 Testovací scénáře 25
4.4 Architektura 30
4.5 Možné přístupy k řešení 31
4.5.1 Implementace vlastního vykreslovacího jádra 32
4.5.2 Využití reálných zařízení 32
4.5.3 Úprava existujícího vykreslovacího jádra 32
Gecko 33
KHTML 33
WebKit 34
Blink 35
4.5.4 Využití kompletní proxy 36
4.5.5 Využití existující funkcionality webových prohlížečů 36
Zabudované ladící nástroje 37
Využití řádkových spouštěcích přepínačů 38
4.5.6 Shrnutí 38

5 Použité technologie během implementace 39
5.1 Chromium Embeded Framework 39
5.2 CEFPython 40
5.2.1 API CEFPythonu 40
Objekty nastavení 41
Rozhraní 41
Provázání s javascriptem 42

6 Postup Implementace 44
6.1 Získání snímku obrazovky 44
6.2 Získání dokumentového objektového modelu webové stránky 44
6.3 Ovlivnění vykreslení 45
6.3.1 Kaskádové styly 45
6.3.2 Javascriptová API 47

6.4 Řešené problémy 49
6.4.1 Překlad ze zdrojových kódů 49
6.4.2 Spouštění instance v prostředí bez zobrazovacího serveru 50
6.4.3 Rychlost vývoje použitých projektů 50

7 Testování 52
7.1 Komentář výsledků testů 52

8 Závěr 62
8.1 Dosažení cílů 62
8.2 Možná budoucí rozšíření 63

Příloha A: Testovací stránka background-size 64

Příloha B: Testovací stránka border-image 65

Příloha C: Testovací stránka box-shadow 66

Příloha D: Testovací stránka opacity 67

Příloha E: Testovací stránka text-shadow 68

Příloha F: Testovací stránka word-wrap 69

Příloha G: Testovací stránka localStorage 70

Příloha H: Testovací stránka sdružená 71

Příloha I: Výsledek testu při povolené localStorage 72

Příloha J: Výsledek testu při nepovolené localStorage 73

Terminologický slovník 74

Použité informační zdroje 75

Rejstřík 79

Seznam obrázků

Obr. 1 Architektura návrhu (Zdroj: autor) 30
Obr. 2 Architektura vykreslovacího jádra Apple WebKit (Zdroj: QUANG, 2015) 34
Obr. 3 Blink a příbuzné komponenty (Zdroj: autor) 35
Obr. 4 Procesy v Chromiu (Zdroj: WILLSON, 2012) 43
Obr. 5 Návaznost projektů (Zdroj: autor) 51

Seznam tabulek

Tabulka 1 Funkční požadavky (Zdroj: autor) 20
Tabulka 2 Aktéři případů užití (Zdroj: autor) 21
Tabulka 3 Případy užití (Zdroj: autor) 21
Tabulka 4 Testovací scénář TC1 (Zdroj: autor) 25
Tabulka 5 Testovací scénář TC2 (Zdroj: autor) 25
Tabulka 6 Testovací scénář TC3 (Zdroj: autor) 26
Tabulka 7 Testovací scénář TC4 (Zdroj: autor) 26
Tabulka 8 Testovací scénář TC5 (Zdroj: autor) 27
Tabulka 9 Testovací scénář TC6 (Zdroj: autor) 27
Tabulka 10 Testovací scénář TJS1 (Zdroj: autor) 28
Tabulka 11 Testovací scénář TJS2 (Zdroj: autor) 28
Tabulka 12 Testovací scénář TI1 (Zdroj: autor) 29
Tabulka 13 Testovací scénář TI2 (Zdroj: autor) 29
Tabulka 14 Testovací scénář TA1 (Zdroj: autor) 30
Tabulka 15 Rozhraní implementovaná v CEFPythonu (Zdroj: TOMCZAK, 2016 [c]) 42
Tabulka 16 Stavy přepínačů funkcionality Blinku (Zdroj: The Chromium authors, 2014 [a]) 47
Tabulka 17 Testování vykreslení (Zdroj: autor) 55
Tabulka 18 Testování DOMu (Zdroj: autor) 57
Tabulka 19 Ověření, zadali dom není zdrojovým kódem (Zdroj: autor) 58
Tabulka 20 Testování komplexního vykreslení (Zdroj: autor) 59
Tabulka 21 Testování alternativního scénáře (Zdroj: autor) 61

1 Úvod

V současné době se velká část aplikací distribuuje ve formě webových aplikací. Jako webové aplikace v této práci budu označovat takové služby, k nímž je očekáván přístup přes http/https protokol z webového prohlížeče a napsané především pomocí technologií HTML CSS a javascript. Takové služby mohou sloužit k mnoha účelům: od objednávání zboží přes komunikaci jednotlivých uživatelů až po řízení firem. Webové aplikace dnes nahrazují ty pro stolní počítače, a už i velcí hráči na trhu vývoje softwaru se orientují tímto směrem. Výhodami pro uživatele webových aplikací jsou především dostupnost z různých geografických lokalit, absence instalace na konkrétní hardware a nezávislost na operačním systému. Pro vývojáře je naopak velkou nevýhodou to, že nezná cílovou konfiguraci, ze které bude uživatel k jeho aplikaci přistupovat.

Webové technologie podléhají mezinárodní standardizaci, ovšem ne všechny webové prohlížeče splňují všechna kritéria. Mnohé techniky se do standardů ještě nedostaly, a u některých nedává smysl, aby je podporovala všechna zařízení.

Přístup na web dnes už dávno není jen z osobních počítačů, ale z mnohem širší škály zařízení. Největší skupinu dnes tvoří mobilní zařízení, hlavně telefony a tablety, ale nezanedbatelnou část tvoří i ostatní typy, například televizory či hodinky. Tento hardware se od sebe mnohdy diametrálně liší v mnoha aspektech, jako jsou velikost zobrazovací plochy, úložná kapacita, výpočetní výkon a podporované technologie. Navíc dle serveru StatCounter návštěvnost z mobilních zařízení překonala návštěvnost ze stolních počítačů v říjnu 2016 (STATCOUNTER, 2016) je proto pochopitelné, že se na ně musí vývojáři soustředit.

Dle výše uvedeného popisu webového prostředí je zřejmé, že toto prostředí je silně heterogenní. Pokud mají webové aplikace sloužit uživatelům skrze všechny tyto konfigurace, je potřeba, aby je vývojář bral na vědomí. Velmi často se z ekonomických důvodů vývoj aplikace soustředí jen na podmnožinu zařízení, a to podmnožinu takovou, která ideálně reprezentuje uživatelskou základnu. I přesto je tato množina tak různorodá, že znemožňuje otestovat aplikaci na všech zařízeních do ní spadajících. Vývojář tudíž neví, zda výsledek u uživatele vypadá tak, jak zamýšlel. I pokud se podaří získat dobrou zpětnou vazbu od uživatele obsahující popis toho, co nefunguje, doprovezený přesnými specifikacemi onoho zařízení, tak vývojář nemusí mít dané zařízení k dispozici a nebude tak schopen situaci odladit.

Důvodem, proč jsem si vybral zpracování tématu je fakt, že pracuji v softwarové firmě, jež se s těmito problémy taktéž potýká.

Tato práce si klade za cíl napomoci řešit zmíněné problémy návrhem softwarového produktu. Stěžejní částí této práce je analýza vývoje softwaru, který ukáže, jak se webová stránka zobrazí na konkrétních zařízeních. Protože práci zpracovávám ve spolupráci

s firmou, pro kterou pracuji, předpokládám, že vývoj a rozvoj softwaru, jenž zpracovávám, bude pokračovat i po odevzdání práce, proto je kladen důraz hlavně na analýzu možných cest, kterými se lze ubírat. Není cílem mít hotové řešení v době odevzdání, které je určené k nasazení. Je preferováno položení základních stavebních kamenů aplikace, kterou bude dále možné rozvíjet, jak se bude objevovat nová funkcionalita prohlížečů a nové webové standardy. Mnohé části práce byly přizpůsobeny kultuře firmy. O přizpůsobení se více rozepisuji v kapitolách 1.2 a 4.1.

1.1 Cíle práce

1.1.1 Hlavní cíle

- Navrhnout software, který ukáže, jak se webová stránka zobrazí na konkrétních zařízeních
- Vybrat vhodné technologie a architekturu pro použití v aplikaci
- Vytvořit prototyp aplikace, který prokáže proveditelnost návrhu

1.1.2 Vedlejší cíle

- Dodat teoretické informace o fungování webových technologií
- Vymezit problém návrhu webových aplikací v kontextu velkého množství a různorodosti zařízení
- Zanalyzovat a kriticky zhodnotit současná řešení snažící se vyřešit tento problém
- Navrhnout, jak zlepšit podmínky vývoje aplikací pro širokou škálu hardwarových konfigurací a prohlížečů
- Zanalyzovat požadavky na aplikaci
- Otestovat prototyp vůči testovacím datům

1.2 Cílová skupina

Tato diplomová práce je zpracovávána ve spolupráci se softwarovou firmou GrandIT s.r.o., která se zabývá vývojem a provozováním softwaru, jak na zakázku, tak vlastního. Firma se pohybuje v oblasti vývoje webového, síťového a mobilního softwaru, mezi jehož klíčové vlastnosti patří vysoká dostupnost, výkon, dodržování standardů a modularita a

nejčastějšími doménami jsou poskytování či agregování nejen multimediálního obsahu a přizpůsobování obsahu klientským zařízením.

Hlavní výstupy této práce využijí v první řadě zaměstnanci firmy k optimalizaci jimi vyvíjených webových aplikací. Nepřímo výsledky práce poslouží všem uživatelům těchto webových aplikací ke zlepšení uživatelského prožitku. Taktéž připadá v úvahu komerční využití, např. pronájem softwaru vývojářům, kteří se taktéž věnují webovému vývoji.

Teoretické informace z této práce mohou využít především ti, jenž se zabývají oblastí fungování prohlížečů a webových aplikací

1.3 Použité metody

Vývoj aplikace probíhá v souladu s metodikou OpenUP, se kterou již mám osobní zkušenosti, a proto v této práci nevybírám metodiku vývoje žádným sofistikovaným způsobem, jakým je například METES (BUCHALCEVOVÁ, 2009). Na základě zkušeností s metodikou ji neaplikuji celou, ale využívám té výhody, že je OpenUP metodikou upravitelnou pro konkrétní potřeby projektu. Specifikem mého projektu je počet členů vývojového týmu (tj. vyjma zákazníků), který je roven jedné. Další výhodou kterou využiji je fakt, že metodika stanovuje pracovní produkty, ale nikoli formu, jak mají vypadat. Proto tyto produkty, pokud jsou relevantní, začleňuji do textu či jiných výstupů; v opačném případě je ponechávám v té formě, ve které mi slouží v průběhu práce.

1.4 Struktura práce

Ve 3. kapitole je analyzován současný stav problematiky. Do té je čtenář uveden v její podkapitole s názvem Popis současného stavu, kde jsou, mimo jiné, shrnuty součásti, ze kterých by se měl systém pro podporu vývoje webových aplikací skládat. Následující část kapitoly se věnuje na trhu již existujícím systémům, jejich výhodám, nevýhodám a využitelnosti pro použití v této práci. Celá kapitola je završena částí, v níž je současný stav shrnut a je zde vysvětleno, proč je vhodné vyvinout nový software.

Klíčovou kapitolou je kapitola 4, která se zabývá analýzou vývoje vytyčené aplikace. V jejích prvních částech jsou probírány úvodní předpoklady práce, následované analýzou uživatelských požadavků, které tuto práci definují. Následuje podkapitola Plán testů, která je stěžejní pro závěrečné zhodnocení, zdali vyvinutá aplikace je v souladu s analýzou.

Dle mého názoru nejzajímavějšími částmi 4. kapitoly jsou její poslední části, kde se analyzuje architektura systému a možné přístupy, jenž je možné využít ve fázi implementace.

Následují kapitoly, které se věnují implementaci softwaru. První z nich představuje technologie, které byly využity v kapite nadcházející. V ní je popsáno, jakým způsobem byl software implementován a jaké v průběhu implementace nastaly problémy.

Kapitola 7. se věnuje testování implementovaného softwaru. Kromě záznamů o vykonaných testech, jsou navíc testy okomentovány a zhodnoceny.

2 Komentovaná rešerše informačních zdrojů

Předpokládám, že hlavním zdrojem, ze kterého budu čerpat, bude práce Václava Pokludy Wapová proxy pro webový obsah (POKLUDA, 2008). Autor se v této práci zabývá analýzou a implementací softwaru, který slouží k optimalizaci a transformaci webových stránek pro účely zobrazení na mobilních telefonech. Prostředí a cíl práce je podobný této. Cílem je pomocí autorem implementovaného softwaru návštěvníkovi webových stránek zprostředkovat co nejlepší možný zážitek při prohlížení na mobilním telefonu. V roce 2008, kdy byla práce napsána, nebylo zastoupení mobilních telefonů na webovém provozu tak silné jako dnes, stejně tak to platí pro výkon těchto zařízení. Webové stránky byly navrhovány s ohledem na očekávané použití na osobním počítači a výsledek v mobilních telefonech tomu odpovídal. Od roku 2008 podíl na trhu i výkon mobilních zařízení vzrostl, s ním však i rozmanitost zařízení. Také já si ve své páci kladu za cíl zkvalitnit uživatelský zážitek na webu, na rozdíl od Pokludovy práce nikoli na výstupu, ale na vstupu. Cílem je implementovat takové řešení, které výrazně zjednoduší vývoj a testování webových aplikací s ohledem na širokou škálu zařízení. Jelikož obě práce mají velmi podobný cíl, domnívám se, že práce Václava Pokludy bude cenným zdrojem.

Zdrojem pro mou práci, který bude významný hlavně v teoretické části je Detecting visually similar Web pages (CHEN, et al., 2010). Autoři se v ní věnují, jak název napovídá, detekcí vizuálně podobných webových stránek. Jejich práce se pohybuje v logické a matematické rovině a podobnost webových stránek řeší z pohledu phishingových útoků (nasměrování uživatele na stránku jinou, ale vizuálně podobnou důvěryhodné, s cílem vylákat z návštěvníka peníze, nebo citlivé údaje). Práce kombinuje několik existujících teorií, z nichž některé mohou být využitelné i pro moji práci. Využití tohoto zdroje v mé práci předpokládám hlavně v oblasti určení metriky podobnosti / ovlivnění uživatelského zážitku některou z neschopností prohlížeče. Podobně jako autorům této práce i mně půjde o určení podobnosti, ačkoli co největší podobnost webových aplikací v mé práci je pozitivním jevem, avšak v práci pánů Chena, Dicka a Millera je tomu naopak.

Práci Device-aware desktop web page transformation for rendering on handhelds (ARTAIL & RAYDAN, 2005) plánuji využít pro čerpání informací týkajících se principů, postupů a technik pro zobrazování webových stránek v mobilních zařízeních. Práce je pro využití jiných než teoretických informací příliš stará a tak hlavním jejím přínosem pro mne budou hlavně poznatky, jak transformace fungují a jaké přinášejí problémy.

Práce Testování webových aplikací za pomoci detekce změn jejího vzhledu (SKELNÁŘ, 2012) přináší zajímavý vhled do testování webových aplikací. Autor vyvinul systém pro porovnání dvou snímků webové aplikace a vizuální analýzy tohoto rozdílu. Bohužel, jeho řešení má ten nedostatek, že vytváří snímky obrazovek z jednoho počítače, čímž zajišťuje stejně velkou vykreslovací plochu u obou obrázků. Nepočítá s responzivním designem ani se změnou rozlišení na konkrétním počítači. Proto tuto práci jakožto zdroj informací nevyužiji

Práce Přístupy k řešení mobilních webových aplikací (NADRCHAL, 2012) se dotýká teorie ohledně responzivního designu (návrhu webových aplikací pro různé velikosti zobrazovací plochy prohlížeče) a jeho hodnocení z pohledu uživatele. Pro mou práci se nehodí, protože se věnuje hlavně samotnému návrhu webových aplikací, rozdílům v očekávání uživatelů webového prohlížeče mobilního a desktopového. Cílem zmíněné práce je především poskytnout obecné rady návrhářům webových aplikací.

Práce Využití HTML5 při vývoji webových aplikací (HANYŠ, 2011) dodává teoretické a praktické informace ohledně technologií z rodiny HTML5. Může posloužit čtenáři mé práce jako vhled do problematiky nových webových technologií, které lze při tvorbě webu použít. Jelikož se jedná o nové technologie, autor poukazuje na fakt, že není tato funcionalita implementována ve všech prohlížečích správně. Má práce se snaží řešit problémy v Hanyšově práci zmíněné, čímž na ni loicky navazuje, avšak mně, jakožto autorovi této práce nové poznatky nepřináší.

Podobně jako já, avšak z druhé strany na problém pohlíží práce Porovnání renderování internetových prohlížečů pro různé operační systémy (MIŠÁK, 2013), která poukazuje na problémy vykreslovaní různými prohlížeči. Práce pouze analyzuje dopady a vybírá prohlížeč s nejlepší podporou technologií. Velkým plusem práce je, že porovnává nejen jednotlivé prohlížeče, ale bere v úvahu i kontext operčních systémů, na který jsou tyto prohlížeče spouštěny. Vzhledem ke stáří práce nemůžu brát prezentované výsledky jako relevantní.

Práce Performance Testing Tool for Web Applications (WARTIAK, 2007)se věnuje problematice testování webových aplikací z pohledu výkonu. Je zde vytyčena problematická doména a předpoklady pro implementaci sofwaru pro tento způsob testování. Mimo jiné jsou v práci analyzována i konkurenční řešení, bohužel, všechna zaměřená na výkonostní testování, nikoli testování funkční. Jde sice hlavně o testování serverových částí webových aplikací, ale i přesto jsou jeho poznatky zajímavé. Já se ve své práci od testování výkonu distancuji a zaměřuji se pouze na testování funkčnosti, a to spíše z pohledu zařízení než aplikací.

Práce Visual Testing of Browser Screen Captures (FRYČ, 2010) využiji v některé další fázi prací na projektu, jehož je tato diplomová práce součástí. Autor se zabývá analýzou měření rozdílu mezi dvěma webovými stránkami z pohledu člověka, nikoli počítače, což považuji za velice originální přístup k problematice. Snímky obrazovky autor sbírá pomocí nástroje Selenium API, tyto snímky následně analyzuje programem vyvinutým v rámci práce. Autor sice tvrdí, že jeho produkt, je možné použít pro testování webových aplikací z různých prohlížečů, nicméně jeho vývoj se soustředil jiným směrem a tak (jak tvrdí v kapitole 8) jeho algoritmus zatím nedává dostatečně dobré výsledky pro mnou zamýšlené použití.

Rozpoznáváním obrázků se detailně zabývá práce Možnosti srovnávání obrázků v mobilních aplikacích (JÍRŮ, 2016). Stejně jako u práce Lukáše Fryče předpokládám

využití v pozdějších fázích vývoje mého softwaru. Na rozdíl od Fryče autorka analyzuje více algoritmů, ty mezi sebou porovnává a vybírá nejvhodnější pro její problematiku. Práci považuji za cennou z hlediska množství poskytnutých teoretických poznatků

Jako jednu z potenciálně nejprospěšnějších prací pro využití v této práci považuji Záznam a přehrávání uživatelských akcích v internetových prohlížečích (KRČMÁRIK, 2016). Ta se zabývá implementací softwaru pro záznam uživatelských interakcí s webovou aplikací. Tato práce nejen že zpracovává část tématu, které je k této práci relevantní, ale zpracovává problém úpravou stávajícího prohlížeče, což je jeden z mnou uvažovaných přístupů.

3 Popis současného stavu

3.1 Pochopení problému

Jak již bylo řečeno v kapitole Úvod, zařízení využívaná pro přístup k webovým aplikacím jsou silně heterogenní, což ztěžuje ladění aplikací. Faktory, které nejvíce ovlivňují vykreslení na daném zařízení, jsou velikost zobrazovací plochy a podpora různých konstrukcí v jazycích používaných na webu. Je ekonomicky náročné testovat aplikace na všech myslitelných zařízeních, a proto je žádoucí nalézt systém, díky kterému bude možné, aby vývojář dostal zpětnou vazbu o tom, jak se webová aplikace zobrazuje na zařízeních z cílové množiny. Takový systém musí mít následující schopnosti:

3.1.1 Určení konkrétního zařízení uživatele

Není příliš velkým problémem zjistit, s jakým zařízením uživatel přichází. Prohlížeč se identifikuje podle takzvané User-Agent http hlavičky, která ve standardizované struktuře předává serveru informaci o tom, přes jaký prohlížeč uživatel na stránku přistupuje. Po analýze tohoto řetězce jsou k dispozici většinou minimálně dvě informace: jaký má uživatel prohlížeč a jaký operační systém. V případě mobilních zařízení je často k dispozici informace o konkrétním typu a značce zařízení.

Tento údaj je jednoduché podvrhnout. Nezabývám se určením zařízení, které se úmyslně identifikuje nepravdivě, ale je žádoucí řešit zařízení, v jejichž identifikaci jsou nepřesnosti či neúplné údaje. Výhodou toho, že je možné User-Agent hlavičku pozměnit je to, že jednoduché ověřit, zdali webová aplikace reaguje na různé typy zařízení různě.

3.1.2 Zjištění schopností daného zařízení

Aby byla uživateli předložena webová aplikace co nejpodobnější formě, kterou navrhl vývojář, je třeba zjistit, jaké schopnosti má prohlížeč uživatele a jaké je možné využít techniky.

Existují dva přístupy ke zjištění podporovaných vlastností prohlížeče uživatele webové aplikace. Jedním z nich je otestování požadovaných vlastností rovnou ve webové aplikaci. Výhodou je, že výsledky jsou přesné (v závislosti na kvalitě provedeného testování), důvěryhodné a aktuální. Nevýhodou je, že takové testování je časově náročné a zdržuje samotnou webovou aplikaci, protože ta čeká na výsledky testování. Další nevýhodou je to, že některá funkcionalita je obtížně automaticky testovatelná.

Druhým přístupem je spolehnout se na nějakou existující databázi. Pro využití takové databáze je třeba mít informace o tom, jaké zařízení a prohlížeč uživatel používá. Po předložení těchto informací databáze vyhledá relevantní data o použitelných technologiích. Výhodou je nižší časová náročnost na získání všech informací a také znalost schopností zařízení ještě před jeho návštěvou webové aplikace. Nevýhodami mohou být zastaralá či neúplná data nebo závislost na externí službě, která nemusí být dostupná. Obecně je tedy problémem důvěryhodnost provozovatele této databáze. Další nevýhodu může být poskytnutí chybných výsledků při nekorektní identifikaci zařízení.

Další možností je zkombinovat oba přístupy; provozovat databázi, která získává data z webových aplikací a to asynchronně tak, aby nezdržovala načtení webové aplikace. Výhodou je získávání čerstvých a přesných dat ze zařízení bez toho, aby zařízení čekala, až bude veškerá jejich funkcionalita otestována. U špatně testovatelných vlastností lze do databáze tato data zanést manuálně. Přetrvávající nevýhodou může být problém při špatné identifikaci zařízení, nevýhodou vlastní databáze může být problémem úzký statistický soubor, to jest kvantita vlastních dat v porovnání s databází, která získává data z více webových aplikací.

3.1.3 Určení cílové množiny

Jak již bylo řečeno výše, z ekonomického hlediska jsou aplikace navrhovány a optimalizovány jen pro určitou množinu zařízení, většinou pro takovou, která reflektuje daný trh a uživatele daných aplikací. U již existujících aplikací jsou sbírána data o přístupech. Ta pak mohou být dále agregována, a na jejich základě lze dělat rozhodnutí, která zařízení podporovat. V kombinaci se zjišťováním schopností zařízení lze rozhodnout, jaké volit technologie a kterým se raději vyhnout.

3.1.4 Vizuální zpětná vazba

Nad daty o tom, jakou funkcionalitou prohlížeč a zařízení disponuje, je možné vytvořit aplikaci, která ukáže, jakým způsobem by se daná webová aplikace zobrazila na onom konkrétním zařízení. Kromě zobrazení vizuálního výsledku lze také získat rozdíl oproti referenčnímu zobrazení. Tuto zpětnou vazbu lze využít jak ve fázi testování webové aplikace, tak ve fázi ladění či implementace. Kromě vykreslení je možné poskytnout i rozdíl v DOM, dokumentovém objektovém modelu webové aplikace, který lépe poslouží vývojáři aplikace. Možnost poskytnout vizuální zpětnou vazbu je naprosto klíčová, protože umožňuje vývojáři nevyvíjet takzvaně „naslepo“, a má možnost si výsledky své práce prohlédnout.

3.1.5 Rozdíl oproti referenčnímu zobrazení

Zmínil jsem, že je možné získat rozdíl oproti referenčnímu zobrazení. Problémem je však to, jak ho definovat. V dnešní době responzivních webových stránek (tj. stránek, které různě vykreslují jednotlivé bloky v závislosti na velikosti obrazovky) a různých javascriptových API (dostupného programového rozhraní) jde o úkol velice náročný, hodný samostatného zpracování. Abstrahuje-li se od výše zmíněného problému, lze zjednodušeně předpokládat referenční zobrazení takové, které je vykresleno v prohlížeči s plně implementovanou podporou všech webových standardů. Nejjednodušší a nejreálnější stanovení referenčního zobrazení, které je v praxi použitelné, je určení, že za vzor je považován výsledek, který vidí vývojář na svém stroji.

Kromě zobrazení rozdílu, je také vhodné určit míru odlišnosti. Na to existují matematické modely, využívané například pro detekci vizuálně podobných stránek (CHEN, et al., 2010) nebo (FRYČ, 2010)

3.1.6 Dodání statistických podkladů

Na základě dat o funkcionalitě prohlížečů a dat z přístupů ke konkrétní webové aplikaci lze udělat statistiku, která odpoví na otázky, kolika uživatelům se aplikace zobrazila nekorektně, která funkcionalita to nejčastěji zapříčinila a, při určení správné metriky, také nakolik byl zážitek uživatele negativně ovlivněn některou z nefunkčností jeho zařízení.

Díky těmto znalostem je pak možné vyvodit příčiny chování skupin uživatelů v aplikaci. Statistiky mohou odpovědět například na otázky, proč nedokončili tito uživatelé objednávku, proč nereagují na nabídky a podobně. Předpokládané odpovědi na tyto otázky jsou většinou toho rázu, že těmto uživatelům se na jejich zařízení špatně zobrazila webová aplikace.

Statistiky o nejproblémovějších technikách je možné rozšířit z kontextu jedné webové aplikace na širší okruh cílových aplikací.

3.2 Analýza existujících řešení

3.2.1 Broo

V současné době firma, pro kterou jsou určeny výstupy této práce (více v kapitole 1.2), využívá sytém s názvem Broo, který slouží pro získávání dat o zařízeních uživatelů přistupujících do webové aplikace. Broo je v první řadě zodpovědné za rozpoznání typu zařízení, modelu zařízení a prohlížeče. Tato data jsou ukládána do tří hierarchických

stromových struktur. U každého z těchto tří znaků tedy platí, že spadá do své kategorie a do všech nadřazených. Například prohlížeč může splňovat znaky Chrome 48, nadřazené Chrome, vyšší WebKit a nejvyšší desktopový prohlížeč. Tato zobecnění umožňují lepší pokládání dotazů, správu databáze, výkon a pochopitelnost údajů. V mnohých případech lze říci, že pokud všechny podřazené kategorie splňují konkrétní vlastnost, jde o vlastnost celé nadřazené kategorie. Tímto způsobem lze získat informace o vlastnostech, které na nejkonkrétnější úrovni nebyly testovány u daného zařízení, to proto, že zařízení spadá do nadřazených kategorií, u nichž tyto vlastnosti známe.

Další výhodou zobecnění je získávání informací z User Agent řetězců (textový řetězec, jímž se prohlížeč identifikuje v rámci požadavku na webovou aplikaci), které mají jen strohé, či neúplné údaje. Ačkoli jsou data v systému Broo uložena ve třech nezávislých stromech, existuje propojení mezi jednotlivými stromy, a díky tomu je možné uložit data o zařízení, které systém nezískal od zařízení přímo, ale má takové znalosti, díky kterým může další data dopočítat. Příkladem může být zjištění mobilní verze prohlížeče Chrome, na základě kterého můžeme vyvodit operační systém Android a na základě doplňujících informací z hlaviček platformu tablet. Předchozí příklad je pouze zjednodušený pro lepší pochopení, ve skutečnosti jsou mechanismy mnohem komplikovanější a zároveň přesnější.

Každá vlastnost prohlížeče, se kterou systém Broo pracuje, má přiřazené testy. Ty jsou dvojího typu: automatické a manuální. Při spouštění automatických testů není třeba interakce s uživatelem. Tento typ testů test se provede sám, vyhodnotí se sám a jeho výsledky se zanesou do systému. Tyto testy jsou použitelné pro testování javascriptových volání a konstrukcí nebo CSS selektorů.

Druhým typem testů jsou testy manuální. Ty mají podobu sady otázek, ve kterých uživatel odpovídá o pravdivosti výroků ohledně vykreslené stránky či reakce prohlížeče. Příkladem je stránka, kde je uživatel dotázán, zda je poskytnutý text opatřen stínem. Dalším příkladem je HTML formulář se vstupními poli, prvním očekávajícím text a druhým očekávajícím telefonní číslo. Uživatel je dotázán, zda se vstupní pole liší. V případě, že se neliší, tak tento typ formulářových vstupů není podporován, ale díky tomu, že novější formulářová pole jsou implementována jako rozšíření stávajících (The World Wide Web Consortium [W3C], 2016) , lze je bezpečně používat a nestává se, že by nebyla přítomna. Nevýhodou tohoto typu testů je, že musejí být spuštěny ručně a vědomě. Jejich vykonání trvá nepoměrně delší dobu a otázky mohou být různými lidmi interpretovány odlišně, stejně tak očekávaný výstup nemusí být osoba schopna vyhodnotit tak, jak vývojář zamýšlel.

Systém Broo v současné době umí zjistit, jakou funkcionalitou zařízení uživatele disponuje, ale neumí dostatečně názorně informovat o tom, co to znamená pro vykreslení konkrétní webové aplikace. Vývojář je nyní omezen na informace typu: konkrétní techniku je možné použít na konkrétním prohlížeči. Díky hierarchickému zobecnění dat je možné učit, že tato technika je použitelná pro celou skupinu prohlížečů či zařízení. Použití systému Broo po jednotlivých technikách, kterými mohou být javascriptová volání, CSS

selektory, HTML atributy či elementy, je ve fázi ladění či testování krajně neefektivní, protože je-li potřeba otestovat všechny použité konstrukce na všech myslitelných prohlížečích a zařízeních, počet testů se rovná kartézskému součinu výše zmíněných faktorů. Prakticky takovéto testování nedává smysl a nepoužívá se.

Plány na rozvoj

Dle plánu budoucího rozšiřování (KOKOŠKA, 2013) je cílem mít možnost z dat systému Broo dodávat agregované výstupy, které reflektují důsledek nepodporované vlastnosti. Takovým výstupem je například otestování všech použitých vlastností a výsledek v konkrétním prohlížeči. Výstup by měl ideálně sestávat z vizuální reprezentace výsledku, seznamu vlastností, které byly použity, ale nejsou podporovány a dodatečných informací, například výsledný DOM. Druhým typem agregovaného výsledku mohou být statistiky ohledně použitelnosti technik a konstrukcí vzhledem ke spektru zařízení.

Takovéto statistiky mohou dávat informaci o tom, na jaké části zařízení je možné tyto vlastnosti využít, nebo naopak které zařízení podporuje nejmenší množinu použitých technik. Toto zařízení se pak jeví jako nejvhodnější pro testování krajních případů.

Poslední plánovanou fází integrace dat ze systému Broo a z konkrétní webové aplikace je využití logu přístupů do webové aplikace pro získání informací o tom, jak se našim uživatelům webová aplikace zobrazuje. Firma přesně ví, jak se na návštěvnosti podílejí jednotlivá zařízení i jak se na nich vykresluje webová aplikace. Tato cenná zpětná vazba vývojářům umožňuje zvolit vhodný způsob, jakým zpříjemnit zákazníkům používání webové aplikace. Díky tomu se můžeme věnovat optimalizaci pouze pro relevantní zařízení a může to mít i další pozitivní dopady na byznys, například dokážeme později interpretovat analýzy chování našich uživatelů díky tomu, že víme, jak se jim aplikace zobrazuje.

3.2.2 WURFL

WURFL je databáze profilů mobilních zařízení a software pro rozpoznávání mobilních zařízeních na základě http hlaviček. V této oblasti je údajně nejpopulárnějším softwarem ve své kategorii (ScientiaMobile, Inc, 2016 [b]). WURFL, podobně jako Broo, reprezentuje data v hierarchické struktuře, které dokáže zobecňovat (ScientiaMobile, Inc, 2016 [a]). Řešení sestává z API pro použití přímo z webu, které je dodáváno pro jazyky Java, Scala, Php a .Net, nebo cloudového API, které dotazuje vzdálenou službu. Další částí je samotná databáze, která může být dle varianty API buď vzdálená, nebo lokální. Další variantou služby je WURFL InFuze, což je modul přímo do web serveru. Výhodou této varianty je především výkon a také efekt, kdy zjištěné vlastnosti jsou dostupné všem webovým aplikacím v rámci http požadavku, a díky tomu se aplikace nemusejí starat o získávání těchto vlastností. Pokud je potřebují, mají je k dispozici.

Nevýhodou tohoto řešení je licenční politika. Zákazník si může vybrat ze dvou přístupů: buď zvolí poměrně restriktivní AGPL licenci, která vyžaduje otevření všech zdrojových kódů (Free Software Foundation, Inc, 2007) propojeného softwaru, nebo zvolí některou z placených variant. Další zásadní nevýhodou je, že WURFL netestuje dané vlastnosti, ale spoléhá se na data dodávaná výrobci zařízení, která jsou často nepřesná a mnohé důležité vlastnosti chybí (KOKOŠKA, 2013). To znemožňuje použití pro detekci moderních technologií, které mnohdy výrobci deklarují za podporované navzdory nekompatibilitě se standardy, nebo technologií, u kterých výrobce nedeklaroval dostupnost vlastnosti, ačkoli ji fakticky zařízení podporuje.

3.2.3 Device Atlas

Device Atlas je řešení velice podobné WURFLu. Také poskytuje lokální i cloudové API v závislosti na tom, zda chce zákazník provozovat databázi u sebe, či dotazovat vzdálenou. V databázi je mnoho údajů, jak o jednotlivých zařízeních a jejich vlastnostech, tak o geografickém rozložení těchto zařízení. API umožňuje zjišťovat, zda se jedná o mobilní telefon, tablet, elektronickou čtečku knih, televizi, webového robota a další. Tato vlastnost, stejně jako mnoho jiných, není dostupná ve verzi Free (Afilias Technologies Ltd, 2016) , která je zdarma. Dalším omezením této verze je možnost provozovat ji pouze v rámci jednoho webu a požadovat detekci maximálně 50 000 x za měsíc. Výrazně širší množinu vlastností a až milion dotazů na detekci umožňuje verze Standard za 40$ měsíčně (Afilias Technologies Ltd, 2016). U lokální databáze není cena přesně určena a záleží tak na konkrétním použití a smlouvě uzavřené mezi zákazníkem a společností Afilias Techologies, která službu Device Atlas provozuje.

3.2.4 Can I use

Can I use je webová aplikace na adrese caniuse.com, která slouží ke zjišťování podpory vlastností napříč prohlížeči. Uživatel vybere jednu z mnoha nabízených vlastností a aplikace mu předloží seznam prohlížečů s úrovní podpory dané vlastnosti. Aplikace též umí seznam prohlížečů vizualizovat dle zastoupení na trhu. Výhodou aplikace Can I use je její jednoduchost a přehlednost v jednoduchých případech užití. Služba je dostupná na adrese caniuse.com.

3.2.5 Caniuse-api

Co činí Can I use aplikaci nepoužitelnou pro naše účely je její interaktivnost a absence API pro strojové získávání a zpracovávání dat, nicméně jeden z autorů Can I use zveřejnil data nasbíraná aplikací Can I use (DEVERIA, 2012-2017) nad nimiž vznikl projekt caiuse-api (BALAYN, 2014), což je javascriptová knihovna pro dotazování dat z Can I use. Knihovna

umí z dat zjistit, zdali je daná vlastnost podporovaná v konkrétním prohlížeči či od které verze a dokáže také zpracovat seznam prohlížečů a rozhodnout, zdali je vlastnost podporovaná všemi. Umí také pro konkrétní vlastnost dodat seznam kompatibilních prohlížečů, které mohou být omezeny relativním zastoupením na trhu nebo počtem verzí starších nebo novějších, než je aktuální stabilní.

Tento projekt je velice zajímavý a vhodný pro využití jakožto inspirace pro implementaci vlastního řešení.

3.2.6 Ostatní webové stránky s přehledem podpory technologií

Stejně jako Can I use, existuje mnoho jiných srovnávacích tabulek s daty o podpoře technologií webovými prohlížeči. Většina z nich používá statická data, popř. jde o komunitou spravované seznamy s ručně zadávanými daty. Většina z nich, kromě Can I use, které je asi nejpovedenějším řešením v dané oblasti, nestojí dle mého názoru za bližší rozepsání.

3.2.7 Webový prohlížeč

Zatím nejvhodnějším nástrojem pro ladění webových aplikací na různých zařízeních je desktopový webový prohlížeč. Google Chrome umožňuje vývojáři přepnout do responzivního módu, buď s vlastním rozlišením obrazovky, nebo s rozlišením jednoho z přibližně dvaceti zařízení. Tam kde to má smysl, umožňuje zobrazení na výšku i na šířku a zobrazení se softwarovou klávesnicí a adresním řádkem. Další emulovanou vlastností v tomto prostředí je rychlost připojení. Chrome umí emulovat všechna běžná mobilní připojení od GPRS přes druhou, třetí a čtvrtou generaci mobilního připojení až po wifi. Volitelně je možné nastavovat odesílanou user agent http hlavičku.

3.2.8 WebPageTest

Na internetu existuje mnoho webových aplikací, které se zaměřují na analýzu webových stránek, mezi nimi například webpagetest.org. Mezi takovýmito řešeními jsem nenašel jediný, který se zaměřuje na technologie použité na stránce a zobrazení z pohledu různých zařízení. Většina se zaměřuje buď na SEO, což je optimalizace pro vyhledávače, nebo na optimalizaci výkonovou. Toto je případ právě webpagetest.org. Test dává na výstupu informace o časové struktuře zobrazení stránky, a v jakém poměru se které technologie časově na zobrazení podílely. Report je rozdělený na aktivity, které se děly (navazování spojení, stahování zdrojů, sestavování DOM a vykreslování) a například u stahování je dostupná statistika, jakou částí se na času stahování podílely obrázky, CSS, javascript a HTML soubory, písma a ostatní. Součástí reportu je také snímek obrazovky.

Výhodou tohoto řešení je možnost vybrat si jeden z určité množiny prohlížečů, z jehož pohledu bude test vykonáván. Zároveň je možné prohlížečům, které to podporují, předat argumenty, které se běžně předávají z příkazového řádku, stejně tak jako povolit či vypnout určité funkce. Webpagetest má veřejně dostupnou dokumentaci, API a dokonce i zdrojový kód pod benevolentní BSD licencí (Open Source Initiative, 1999). Architektura tohoto řešení je modulární, takže je možné se některými částmi inspirovat a některé převzít.

S tímto projektem souvisí software mobitest-agent, což je nezávisle vyvíjená součást projektu webpagetest sloužící k testování na mobilních zařízeních. Dodává webpagetestu možnost testovat mobilní verze prohlížečů. Problematické ovšem je, že WebPageTest je navržen pro testování na reálných zařízeních, která jsou připojena k řídícímu počítači, který testy řídí, avšak testy samotné neprovádí (WebPagetest comunity, 2014). Tento projekt je silně motivován měřením výkonu webových aplikací a nikoli funkčním testováním, proto tento projekt není pro mé použití vhodný.

3.3 Obhajoba vlastního řešení

Zásadním nedostatkem současných řešení, je fakt, že v sobě neintegrují všechnu funkčnost, která je zmíněna v kapitole 3.1 Pochopení problému. Každé z těchto produktů řeší pouze některou část problému.

Druhým silným nedostatkem, kterým trpí software třetích stran, tj. nikoli interně vyvíjené Broo, je nedůvěryhodnost dat. Vzhledem k tomu, že se nemůžeme spolehnout na způsob, jak výše zmíněné produkty získávají data, nemůžeme se spolehnout na data samotná. A jelikož chceme provozovat kvalitní službu, musíme mít kvalitní data. S tím souvisí další aspekt, a to omezená přizpůsobitelnost. Jelikož se technologické prostředí webu rychle mění, je potřeba mít možnost rychle a pružně reagovat na požadavky trhu a toho můžeme dosáhnout jen s produktem, který můžeme přizpůsobit. Žádné ze zmíněných řešení pro detekci podpory technologií neumožňuje přidání nových vlastností, které si přejeme sledovat.

Některé z produktů, např. DeviceAtlas, umožňují napojení na konkrétní web a tím počítat statistiky relevantní množině přistupujících uživatelů. Nicméně licenční omezení neumožňují provozovat DeviceAtlas zadarmo a zároveň nad více webovými aplikacemi.

Co žádný z výše uvedených produktů neumožňuje, je poskytnutí vizuální zpětné vazby a analýza vlastností použitých ve webové aplikaci, které jsou problematické v konkrétním zařízení. Toto je klíčové pro ladění aplikací samotnými vývojáři bez nutnosti dané zařízení vlastnit.

Vzhledem k nedostatkům existujících řešení se ukazuje, že je nutné vyvinout modul, který bude schopen poskytnout vizuální zpětnou vazbu. Z byznysového hlediska je pro firmu

výhodnější rozvoj jejího stávajícího řešení s názvem Broo. Jeho výhodou je plná kontrola nad daty a jejich získáváním a z toho plynoucí důvěryhodnost, dále také možnost ovlivnit a přizpůsobit systém vlastním potřebám a potřebám zákazníků. Z dlouhodobého pohledu ale považuji za vhodné, aby byla vyvinuta komponenta pro zobrazování, která bude nezávislá na službě pro získávání dat o schopnostech prohlížečů. To nejenže je v souladu s myšlenkou komponentového vývoje aplikací, ale pokud by se jiná služba ukázala jako vhodnější, bude jednodušší vyměnit pouze tuto část a ponechat jádro.

4 Analýza vývoje softwaru

4.1 Vstupní předpoklady práce

Poněvadž má tato práce sloužit firmě GrandIT, musím respektovat mnohá, i nevyřčená, omezení, jejichž dodržení zajistí, že firma dostane maximální přidanou hodnotu z této práce. Jde hlavně o omezení technologická, ale zanedbatelná nejsou ani ta ostatní. Účelem těchto omezení je hladká integrace do stávající softwarové základny, podnikové kultury, a technického prostředí s ohledem na budoucí vývoj.

4.1.1 Technické předpoklady

V této práci je cílem rozvoj současného softwaru pro získávání informací o zařízení přicházejícím do webové aplikace, proto musím respektovat jeho technické předpoklady.

Většina softwaru vyvíjeného ve firmě GrandIT je rozdělena na komponenty. Základní rozdělení rozlišuje frontend a backend: Frontendem se rozumí kód, který slouží k prezentaci dat všem uživatelům a interakci s nimi. Na rozdíl od některých (Adaptic, s.r.o., 2005) naše chápání frontendu se neomezuje pouze na administrační část webu. Dále je vhodné říci, že frontendem označujeme nejen kód na straně klienta, ale i mnohý kód serverový. (Připodobněním ke známému architektonickému vzoru Model-View-Controller je rolí frondendu obsluha komponent View a Controller.) Backendem označujeme komponentu, která má na starost výkonný kód, tj. komunikaci s databází, popřípadě s backendy ostatních služeb, řízení a vykonávání potřebných výpočtů atp. K backendu může být připojena široká škála služeb a frontendů a to i současně. Backend přijímá a vystavuje data ve standardní struktuře a je na frontendu nezávislý. V této práci implementuji výhradně backend aplikace a frontendem se zde nezabývám.

Backendy jsou postaveny na architektuře mikroslužeb. Jedná se o koncept, kdy se software skládá z jednotlivých služeb. Každá služba je zodpovědná za jednu z doménových oblastí a běží ve vlastním procesu nezávisle na ostatních, často na jiném serveru, který může být i v jiné geografické lokalitě. Mikroslužby obecně navzájem komunikují předáváním zpráv ve standardní struktuře přes síť (FOWLER, 2014). Ve firmě se v současné době pro implementaci služeb využívá techniky RPC. RPC znamená v angličtině remote procedure call, volně přeloženo jako vzdálené volání metod. Jde o volání metod na vzdáleném serveru přes http nebo https či spdy protokol, nečastěji ve formátu JSON, ale protože přes http protokol mohou projít libovolná data za předpokladu, že jsou správně popsána http hlavičkami, mohou metody vracet jakkoli strukturovanou odpověď. Komunikace probíhá tak, že jedna služba vytvoří http požadavek, který je většinou textovým řetězcem a ten

následně pošle na vzdálený server na port, kde služba poslouchá. Vzdálená služba požadavek zpracuje a vrátí odpověď v odpovídajícím formátu. Výhodou komunikace přes http je jednak standardizace a jednak možnost komunikovat se službou přes běžný webový prohlížeč, což zjednodušuje jak integraci různých služeb různých vývojářů, tak ladění.

Aktuálně se ve firmě služby programují v jazyce Python ve verzi 2.7 a využívá se vlastních knihoven pro vytváření RPC serverů a klientů. Aplikace v Pythonu jsou na http rozhraní vystaveny pomocí WSGI rozhraní, které se stará o komunikaci mezi webovým serverem a aplikací. Servery firmy běží převážně na GNU/Linuxové distribuci Debian ve verzi 8,0 Jessie, která aktuálně nahrazuje starší verzi 7.0 Wheezy

4.1.2 Požadavky na zdrojový kód

Zdrojový kód v jazyce Python se řídí oficiálním standardem PEP8 (ROSSUM, et al., 2001), který předepisuje formátování zdrojového kódu. Výhodou používání standardního formátování je lepší čitelnost, přehlednost a proniknutelnost do neznámého zdrojového kódu. Jelikož Python je multiparadigmatický programovací jazyk, lze v něm používat různé konstrukce pro dosažení stejného výsledku. Zavedenými a akceptovanými paradigmaty ve firmě jsou objektové a funkcionální programování, případně jejich smysluplná kombinace.

Ideálním stavem ve zdrojovém kódu je minimum závislostí na softwaru třetích stran. Pokud je nutné nějakou závislost začlenit, nejvhodnější způsob je použití pythonního balíčkovacího systému PIP. Důležitým faktorem u cizího softwaru je jeho licence. Protože zatím není v plánu vydání softwaru, který v rámci této práce rozšiřuji, jako open-source, je nutné volit software s takovými licencemi, které nevyžadují zveřejnění zdrojového kódu.

U přijímání cizího softwaru je také nutné brát v úvahu ještě další faktory. Výkon, kvalita a paměťová náročnost jsou třemi z nich. Pokud je software kvalitní, výkonný, paměťově nenáročný, což znamená bez defektů, a vyhovující současným a budoucím požadavkům, pak se již dalšími faktory nemusíme zabývat. Ovšem software ve většině případů takové vlastnosti nemá, a tak je nutné při přijímání cizího softwaru zohlednit následující: zda je software stále ještě vyvíjen, jsou včas opravovány defekty a software je přizpůsobován novým podmínkám. U softwaru s takovými vlastnostmi máme alespoň částečnou jistotu, že v budoucnu nebudeme muset cizí kód udržovat vlastními silami.

U softwaru, který již není vyvíjen, je vhodné při jeho zapojování do vlastní infrastruktury zohlednit jeho udržovatelnost a komplexitu, a to hlavně z důvodu co nejjednoduššího budoucího vývoje, oprav a rozšiřování produktu.

4.2 Analýza uživatelských požadavků

V souladu s cíli práce a informacemi v úvodu je třeba analyzovat požadavky na aplikaci, která je navrhována. V úvahu je třeba brát potenciální uživatelskou základnu, zacílení softwaru a strategické plány firmy do budoucna.

4.2.1 Současné působení firmy

Vizí firmy je vytvářet a dodávat aplikace optimalizované tak, aby jejich užívání bylo pohodlné na celém spektru zařízení a to jak z hardwarového, tak softwarového hlediska. V referencích firmy lze, kromě nativních aplikací pro různá zařízení, nalézt především aplikace webové, které tvoří největší část dodávaných produktů, ale také aplikace pro hybridní televizní a kabelové vysílání, takzvané HbbTV aplikace. Technologie HbbTV představuje možnost zkombinovat výhody televizního vysílání a surfování na webu a tím posilnit uživatelský zážitek. Relevantní částí standardu HbbTV je fakt, že aplikace provozované na platformě HbbTV jsou implementovány jako webové aplikace, které interagují s televizním vysíláním. Firma GrandIT aktuálně vidí největší potenciál právě v tomto druhu aplikací a má již s jejich vývojem zkušenosti. Ty například ukazují, že hardware i software určený k běhu televizních aplikací bývá často poddimenzovaný a nezdokumentovaný. Využívané prohlížeče často nepodporují techniky, které jsou již několik let součástí webových standardů a tak vývoj pro tuto platformu musí být ještě opatrnější než vývoj cílený na uživatele chytrých telefonů. Podobnost vývoje pro mobily i televize je v tom, že na trhu existuje velké množství produktů s rozličnými konfiguracemi, a proto je třeba vyvíjet s ohledem na technologie podporované těmito konkrétními produkty. Ovšem výhodou je již zmíněný fakt, že se jedná o webové aplikace a proto testování a analýza těchto aplikací může být stejná, jako v případě běžných webových aplikací.

4.2.2 Cíle firmy

Dle rozhovorů s vedoucím oddělení vývoje softwaru, Lukášem Kokoškou, je jedním z prostředků dosažení vize vývoj a rozvoj vlastního řešení, které zajistí lepší kompatibilitu firmou vyvíjených aplikací se zařízeními uživatelů přistupujících k těmto aplikacím. Aktuálně je k sytému pro rozpoznávání prohlížečů a hardwaru uživatelů připojen například image server. To je komponenta, která na základě znalosti hardwarového profilu škáluje obrázky předkládané uživateli na adekvátní velikost a kvalitu. To má za následek nižší datové přenosy a z toho plynoucí rychlejší načítání, které pozitivně ovlivňuje uživatelský zážitek. Do budoucna je v plánu rozšíření portfolia podobných komponent a využití dat nasbíraných za dobu provozu systému Broo. Hlavním směrem je využití těchto dat k zajištění vývoje kvalitnějších a optimalizovanějších aplikací.

Firma si přeje mít způsob testování kompatibility webových aplikací s hardwarem a softwarem uživatelů přistupujících k těmto aplikacím, aniž by vývojáři museli tato zařízení vlastnit. Do budoucna chce také mít ex-post analýzu, která bude schopná z dat o provozu aplikace podat souhrnné reporty o kvalitě zobrazení na klientském hardwaru.

4.2.3 Funkční požadavky

Tabulka 1 ukazuje, jaké funkční požadavky na software uživatelé mají. Uživateli je v této kapitole myšlena skupina vývojářů, kteří ladí danou webovou aplikaci pomocí mého softwaru. Funkční požadavky určují, jaká funkcionalita je od softwaru očekávána. Sloupec priorita udává, jakou důležitost uživatelé dané funkcionalitě přisuzují. Posledním sloupcem je odkaz na případy užití, které uvádí Tabulka 3.

Tabulka 1 Funkční požadavky (Zdroj: autor)

Id	Název požadavku	Popis	Priorita *(1 vysoká 3 nízká)*	Případ užití
F1	Získání vizuální podoby stránky	Uživateli je po zadání URL stránky a identifikace zařízení dodán obrázek zobrazující vykreslení webové stránky	1	UC1
F2	Získání DOMu stránky	Uživateli je vrácena reprezentace dokumentového objektového modelu stránky	1	UC1
F3	Získání chyb funkčnosti	Uživateli je po zadání série interakcí s webovou aplikací dodán seznam problémů, ke kterým nastalo	2	UC2
F4	Získání statistik chyb z logů návštěvnosti	Uživateli je předložena statistika za konkrétní časové období, kde je zobrazeno, které části webové aplikace dělají proporcionálně nejčastější problémy na zařízeních uživatelské základny	3	UC3

4.2.4 Případy užití

Tabulka 3 ukazuje případy užití, které vycházejí z funkčních požadavků na aplikaci. Nepovažuji za nutné zde prezentovat celý diagram případů užití včetně obrázků a kompletních tabulek. V mém projektu jsou případy jednoduché, a proto jsem zvolil tuto jednoduchou formu, která udává název a cílový stav. Tabulka 2 udává, jací aktéři ve výše zmíněné tabulce vystupují.

Tabulka 2 Aktéři případů užití (Zdroj: autor)

Aktér (Actor)	Popis
Vývojář webové aplikace	Osoba nebo tým, který kóduje novou, či upravuje stávající webovou aplikaci.
Provozovatel webové aplikace	Osoba, jejímž cílem je maximalizovat uživatelský komfort při používání aplikace. Může to být například subdodavatel, vývojářská firma nebo manažer firmy poptávající webovou aplikaci.

Tabulka 3 Případy užití (Zdroj: autor)

Id	Název	Cílový stav
UC1	Ověření vykreslení	Poptávající aktér ví, jak vypadá jím požadovaná vykreslená stránka.
UC2	Ověření funkčnosti	Poptávající aktér ví, jak webová aplikace zareaguje na jím definované akce.
UC3	Získání statistik	Poptávající aktér má report o problémech s vykreslením a funkčností webové aplikace za dobu provozu, či specifickém časovém intervalu.

4.2.5 Uživatelské scénáře

Uživatelské scénáře vycházejí z funčních požadavků s prioritou jedna. Všechny zmíněné scénáře jsou součástí případu užití s identifikátorem UC1. Z nich následně vychází plán testů, kde je vše blíže rozepsáno.

Získání vizuální podoby stránky

Hlavní scénář

- Uživatel zadá User-Agent a URL existující stránky
- Systém vrátí vykreslenou stránku v podobě obrázku

Alternativní scénář, kdy stránka nemůže být načtena, např. kvůli špatnému URL

- Uživatel zadá User-Agent a URL neexistující stránky
- Systém vrátí prázdný obrázek o správných rozměrech

Získání DOMu webové stránky

Hlavní scénář

- Uživatel zadá User-Agent a URL existující stránky
- Systém vrátí řetězec reprezentující dokumentový objektový model stránky

Alternativní scénář, kdy stránka nemůže být načtena, např. kvůli špatnému URL

- Uživatel zadá User-Agent a URL neexistující stránky
- Systém vrátí prázdný řetězec

Získání chyb funkčnosti

Hlavní scénář:

- Uživatel zadá User-Agent a URL existující stránky a sérii platných akcí se stránkou (klepnutí na prvek, posun stránky, klepnutí na klávesnici) a požadovaný výstup (DOM, nebo vizuální podobu)
- Systém vrátí uživateli DOM, nebo vizuální podobu stránky a informaci, že vše proběhlo v pořádku

Alternativní scénář, kdy jedna z akcí nemůže být dokončena (například chybí prvek pro klepnutí):

- Uživatel zadá User-Agent a URL existující stránky a akci, kterou nelze provést (klepnutí na neexistující prvek)
- Systém sdělí uživateli, kterou akci nebylo možné dokončit a dodá DOM a vizuální podobu stránky v době, kdy se nepodařilo akci dokončit.

4.2.6 Nefunkční požadavky

Nefunkční požadavky se řídí poznatky z kapitoly 4.1 Vstupní předpoklady práce. Nejedná o kriticky důležitý systém, a navíc jde pouze o jeho backendovou část. Proto v této části bude jen stručné shrnutí požadavků, které jsou bezpodmínečně nutné.

- Software musí fungovat na GNU/Linuxové distribuci Debian ve verzi Jessie
- Software musí umět komunikovat se současným rozpoznávačem Broo
- Software musí být udržovatelný a jednoduše pochopitelný pro případné další vývojáře

4.3 Plán testů

Cílem této kapitoly je definovat, jaké testy na úrovni projektu je třeba stanovit. Jelikož je jedním z cílů dodat prototyp aplikace, věnují se plány testů právě tomuto prototypu. Z toho důvodu je třeba stanovit testy jednak obecněji a jednak s ohledem na to, že vyprodukovaný software je prototypem a nikoli produktem určeným k okamžitému nasazení. Z toho důvodu zde neplánuji akceptační testy tak, jak je obvyklé, ale soustředím se hlavně na testy funkční a integrační. Stejně tak nebudou v tomto textu řešeny testy jednotek, protože prototyp se bude v průběhu vývoje rychle měnit a nebude tak komplexní, aby se implementace jednotkových testů vyplatila. Vzhledem k použitým technologiím (Kapitola 5) je automatizace těchto testů komplikovaná.

Cílem testů je ověřit implementovanou funkcionalitu softwaru. Plán testů slouží k jejich definování, a je rozepsán dále v této kapitole. Předmětem testování je jednak implementovaná funkcionalita, ale i její části. Testy vycházejí z funkčních požadavků s prioritou 1 a nefunkčních požadavků.

4.3.1 Testovací prostředí

Pro funkční testy byly vytvořeny jednoduché webové stránky vystavené na lokálním http serveru. Tyto stránky využívají technik a schopností prohlížečů, jejichž funkcionalita je testována. Pro každou testovanou vlastnost je vytvořena jedna stránka, která je zaměřena specificky na tuto vlastnost. Dále existuje stránka, která využívá kombinace těchto vlastností.

4.3.2 Testovací nápady

Pro otestování správně implementovaného ovlivnění podpory kaskádových stylů byly vybrány vlastnosti, které umí otestovat rozpoznávací systém Broo, konkrétně

- Vlastnost background-size
- Vlastnost border-image
- Vlastnost box-shadow
- Vlastnost opacity (neplést s barevnou paletou RGBA)
- Vlastnost text-shadow
- Vlastnost word-wrap s hodnotou break-word

Jelikož u těchto vybraných CSS vlastností nedochází v případě jejich nepodpory ke změně textové reprezentace dokumentového objektového modelu, není proto třeba ji testovat,

protože ta bude stále stejná. Naopak u javascriptu je nevyhnutelné DOM otestovat, a to nejen zda bude ovlivněn, ale také jakým způsobem. Cílem je dosažení stavu identického se stavem na reálném zařízení, které dané technologie nepodporuje. V případě javascriptu je testován také stav kontextu, ve kterém se konkrétní testovaná technologie vyskytuje. Například efekt volání

```
localStorage.setItem("key", "value");
```

je v případě nepodpory technologie localStorage odlišný od nezavolání této konstrukce. V prvním případě totiž dojde k vystřelení výjimky v daném kontextu, a další příkazy, které následují, nejsou vykonány. Naopak při odstranění či nezavolání konstrukce je následující programový kód vykonáván dále.

Pro testování javascriptu byla vybrána technologie localStorage, která umožňuje ukládání dat na klientském zařízení. Tato technologie byla vybrána hlavně proto, že s její nepodporou jsme se ve firmě už potýkali a zásadně ovlivnila návrh webové aplikace. Dalším faktorem, proč byla vybrána, je její jednoduchá testovatelnost, a to jak výsledného vykreslení, tak dokumentového objektového modelu.

Pro otestování, zda software umí ovlivnit vykreslení a sestavení DOMu, je každá vlastnost nejdříve testována samostatně se zapnutou a s vypnutou podporou. Poté je otestován případ, kdy cílová stránka využívá všech technologií otestovaných samostatně. Tento případ je pak otestován z pohledu dvou zařízení, jednoho s plnou podporou a jednoho s částečnou. Současně je testováno, zda je výsledný vykreslený obrázek ve správné velikosti. U testování DOMu je také třeba ověřit, zda není vypisován pouhý zdrojový kód stránky, ale skutečná textová reprezentace dokumentového objektového modelu včetně změn provedených javascriptem.

4.3.3 Testovací scénáře

Tabulka 4 Testovací scénář TC1 (Zdroj: autor)

Id	TC1	
Název	Ovlivnění podpory CSS box-shadow	
Účel	Ověřit, zda software umí zapínat a vypínat podporu CSS vlastnosti box-shadow.	
Kroky	1	Tester zapne podporu box-shadow.
	2	Tester poptá skrze software webovou stránku využívající tuto technologii.
	3	Systém vrátí obrázek s vykreslenou stránkou.
	4	Tester vypne podporu box-shadow.
	5	Tester zopakuje kroky 2 a 3.
Očekávaný výsledek	V prvním obrázku je u elementu, v jehož stylu je použita vlastnost box-shadow, zobrazen stín. Ve druhém obrázku je tento element beze stínu.	

Tabulka 5 Testovací scénář TC2 (Zdroj: autor)

Id	TC2	
Název	Ovlivnění podpory CSS background-size	
Účel	Ověřit, zda software umí zapínat a vypínat podporu CSS vlastnosti background-size. Neslouží k ověření backround-repeat, tj. kontrole opakování obrázku na pozadí.	
Kroky	1	Tester zapne podporu background-size.
	2	Tester poptá skrze software webovou stránku, na níž je element, který má na pozadí fotografii, zmenšenou pomocí CSS background-size.
	3	Systém vrátí obrázek s vykreslenou stránkou.
	4	Tester vypne podporu background-size.
	5	Tester zopakuje kroky 2 a 3.
Očekávaný výsledek	V prvním obrázku má element, v jehož stylu je použita vlastnost background-size, na pozadí menší fotografii, než je na obrázku druhém. Fotografie na pozadí se může a nemusí opakovat.	

Tabulka 6 Testovací scénář TC3 (Zdroj: autor)

Id	TC3	
Název	Ovlivnění podpory CSS border-image	
Účel	Ověřit, zda software umí zapínat a vypínat podporu CSS vlastnosti border-image. Není cílem ověřit border-image-slice ani border-image-repeat, protože rozpoznávač Broo řeší jen obecnou podporu vlastnosti border-image.	
Kroky	1	Tester zapne podporu border-image.
	2	Tester poptá skrze software webovou stránku, na níž je element s obrázkovým okrajem implementovaným přes border-image a bez jakéhokoli jiného okraje.
	3	Systém vrátí obrázek s vykreslenou stránkou.
	4	Tester vypne podporu border-image.
	5	Tester zopakuje kroky 2 a 3.
Očekávaný výsledek	V prvním obrázku má element, v jehož stylu je použita vlastnost border-image, zvýrazněný okraj, ve druhém obrázku je zcela bez okraje.	

Tabulka 7 Testovací scénář TC4 (Zdroj: autor)

Id	TC4	
Název	Ovlivnění podpory CSS vlastnosti opacity	
Účel	Ověřit, zda software umí zapínat a vypínat podporu průhlednosti skrze CSS vlastnost opacity. Nejde o podporu barevné palety RGBA ani poloprůhledných PNG obrázků s alfa kanálem.	
Kroky	1	Tester zapne podporu vlastnosti opacity.
	2	Tester poptá skrze software webovou stránku, na níž jsou přes sebe dva elementy, přičemž hornímu elementu je nastavena výrazná barva pozadí a sytost / průhlednost o hodnotě blízké polovině intervalu [0; 1].
	3	Systém vrátí obrázek s vykreslenou stránkou.
	4	Tester vypne podporu vlastnosti opacity.
	5	Tester zopakuje kroky 2 a 3.
Očekávaný výsledek	V prvním obrázku je skrze horní element vidět obsah spodního elementu. Ve druhém obrázku má pozadí horního elementu plnou sytost a není skrze něj vidět.	

Tabulka 8 Testovací scénář TC5 (Zdroj: autor)

Id	TC5	
Název	Ovlivnění podpory CSS vlastnosti text-shadow	
Účel	Ověřit, zda software umí zapínat a vypínat podporu CSS vlastnosti text-shadow.	
Kroky	1	Tester zapne podporu vlastnosti text-shadow.
	2	Tester poptá skrze software webovou stránku, na níž je text, u kterého je nastaven stín.
	3	Systém vrátí obrázek s vykreslenou stránkou.
	4	Tester vypne podporu vlastnosti text-shadow.
	5	Tester zopakuje kroky 2 a 3.
Očekávaný výsledek	V prvním obrázku je text opatřen stínem, ve druhém nikoli.	

Tabulka 9 Testovací scénář TC6 (Zdroj: autor)

Id	TC6	
Název	Ovlivnění podpory CSS vlastnosti word-wrap s hodnotou break-word	
Účel	Ověřit, zda software umí zapínat a vypínat podporu lámání dlouhých slov CSS hodnotou break-word u vlastnosti word-wrap. Nejde o podporu vlastnosti word-break, text-overflow ani hyphens.	
Kroky	1	Tester zapne podporu CSS vlastnosti word-wrap a její hodnoty break-word.
	2	Tester poptá skrze software webovou stránku, v níž je text obsahující slovo, které je delší, než je šířka elementu, v němž je text obsažen.
	3	Systém vrátí obrázek s vykreslenou stránkou.
	4	Tester vypne podporu vlastnosti word-wrap a její hodnoty break-word.
	5	Tester zopakuje kroky 2 a 3.
Očekávaný výsledek	V prvním obrázku je dlouhé slovo rozděleno tak, aby žádný znak nepřetekl přes pravý okraj elementu. Ve druhém obrázku je toto slovo vcelku a pokračuje přes pravý okraj elementu.	

Tabulka 10 Testovací scénář TJS1 (Zdroj: autor)

Id	TJS1	
Název	Ovlivnění podpory javascriptové technologie localStorage	
Účel	Ověřit, zda software umí správně emulovat (ne)podporu localStorage v javascriptu, a zachovat se korektně ke kontextu, v němž je některá z metod, či vlastností localStorage volána.	
Kroky	1	Tester zapne podporu localStorage.
	2	Tester poptá skrze software webovou stránku, která používá textová data z localStorage a vykresluje je na obrazovku, a dále vypisuje i další data.
	3	Systém vrátí obrázek s vykreslenou stránkou a její DOM.
	4	Tester vypne podporu localStorage.
	5	Tester zopakuje kroky 2 a 3.
Očekávaný výsledek	První výsledek, a to jak DOM, tak obrázek, obsahují text získaný z localStorage a další texty, které byly ve skriptu vypisovány. Druhý výsledek neobsahuje data, která jsou v prvním případě získána z localStorage, ani další text vypisovaný v rámci skriptu. Texty vypisované v jiném kontextu jsou korektně zobrazeny.	

Tabulka 11 Testovací scénář TJS2 (Zdroj: autor)

Id	TJS2	
Název	DOM není jen zdrojový kód	
Účel	Ověřit, zda výsledný DOM stránky je skutečnou reprezentací DOMu se všemi elementy po vykonání všech změn a nikoli jen zdrojovým kódem stránky.	
Závisí na	TJS1	
Kroky	1	Tester vykoná test TJS1.
	2	Systém během testu TJS1 vrátí dvakrát DOM stránky.
Očekávaný výsledek	DOM v obou případech obsahuje nejen zdrojový kód stránky, ale i elementy vytvořené JavaScriptem.	

Tabulka 12 Testovací scénář TI1 (Zdroj: autor)

Id	TI1	
Název	Software vrátí obrázek ve velikosti, kterou k danému zařízení deklaruje Broo	
Účel	Ověřit, zda software vrací obrázek s takovými rozměry, které byly získány z rozpoznávacího systému Broo	
Kroky	1	Tester v provozní konfiguraci softwaru poptá webovou stránku pro zařízení, pro nějž systém Broo zná rozlišení obrazovky.
	2	Systém vrátí obrázek vykreslené stránky.
	3	Tester v systému Broo zjistí deklarovanou velikost.
Očekávaný výsledek	Výsledný obrázek má takovou velikost v pixelech, jakou udává Broo pro toto zařízení.	

Tabulka 13 Testovací scénář TI2 (Zdroj: autor)

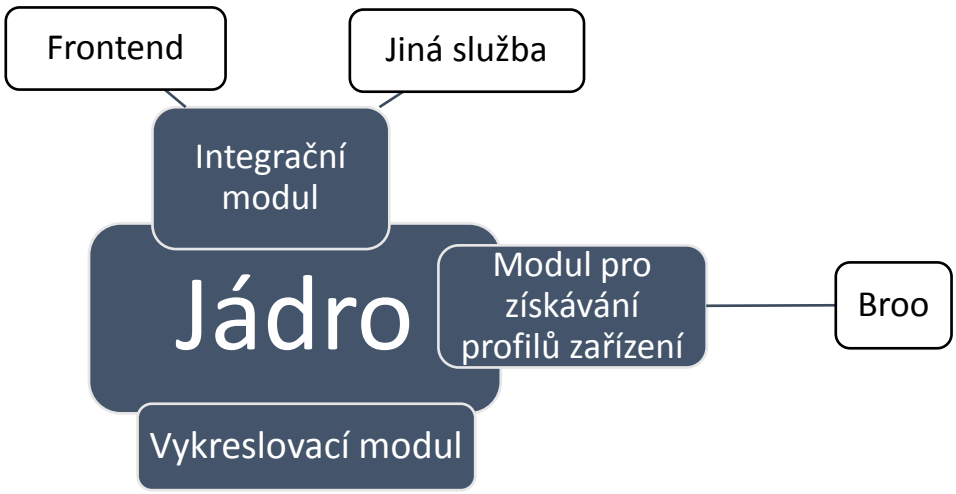
Id	TI2	
Název	Software umí dodat komplexní pohled na vykreslenou stránku	
Účel	Ověřit, zda software umí využít svých schopností najednou a dodat obrázek vykreslené stránky, která používá více než jednu vlastnost či technologii z testů TC1 – TC6, TJS1 a zároveň splňuje kritéria testovacího scénáře TJS2 a TI1.	
Závisí na	TC1, TC2, TC3, TC4, TC5, TC6, TJS1, TI1	
Kroky	1	Tester v provozní konfiguraci softwaru poptá webovou stránku, která používá více technologií a vlastností z testů, na kterých tento test závisí. Tuto stránku poptá pro zařízení, pro nějž systém Broo zná rozlišení obrazovky, a které alespoň jednu vlastnost nepodporuje.
	2	Systém vrátí obrázek vykreslené stránky a DOM stránky.
	3	Tester stejnou stránku poptá pro zařízení s jinou velikostí obrazovky a jinou množinou podporovaných vlastností.
	4	Systém vrátí obrázek vykreslené stránky a DOM stránky.
	5	Tester z Broo zjistí data o velikostech obou zařízení v pixelech a podporované funkcionalitě.
Očekávaný výsledek	Oba obrázky mají takovou velikost, jaká je v Broo uvedena. Obrázek výsledné stránky byl vykreslen s použitím právě těch technologií, které zařízení dle Broo podporuje.	

Tabulka 14 Testovací scénář TA1 (Zdroj: autor)

Id	TA1	
Název	Software se zachová korektně při nenačtení stránky	
Účel	Ověřit, zda software zvládne situaci, kdy se nepovede načíst stránku, dle scénáře případů užití.	
Kroky	1	Tester v provozní konfiguraci softwaru poptá webovou stránku, která neexistuje.
	2	Systém vrátí obrázek a DOM.
Očekávaný výsledek	Bílý obrázek má takovou velikost, jaká je v Broo uvedena, DOM je prázdný řetězec a program je připraven vykonávat další kód.	

4.4 Architektura

Jedním z požadavků na software je jeho rozšiřitelnost, a to hlavně z důvodu toho, že je vyvíjen jako prototyp, na kterém bude dále pracováno. Proto navrhuji architekturu hodně obecnou a definuji v ní základní komponenty, které v tomto softwaru v době jeho návrhu očekávám. Na Obr. 1 Architektura návrhu (Zdroj: autor) je zobrazen systém tak, aby odrážel veškeré požadavky stanovené výše.



Obr. 1 Architektura návrhu (Zdroj: autor)

Tmavé pozadí mají nově implementované komponenty, kterými se v práci věnuji, kdežto pozadí světlé značí externí komponenty softwaru, jejichž návrhem se nezabývám. Jde o

komponenty, které ještě ani nemusí existovat. Mezi ně počítám frontendy, jejichž účelem bude uživatelsky přívětivým způsobem zprostředkovávat služby této aplikace. Dále sem patří komponenty, které v budoucnu budou chtít využít funkcionality tohoto softwaru. Na straně druhé je externí komponenta již existující, konkrétně rozpoznávací systém Broo.

Mezi ním a jádrem systému je tenká vrstva zodpovědná za komunikaci a překlad dat do požadované formy. Jádro je část aplikace, která slouží pouze k integraci jednotlivých modulů a zajištění komunikace mezi nimi. Samotné jádro neobsahuje žádnou složitou výpočetní logiku. Nejdůležitější komponentou je Vykreslovací modul. To je klíčová komponenta celého systému, které je věnována v rámci implementace i technologické architektury největší pozornost. Je zodpovědná nikoli pouze za vykreslování, ale také za veškeré získávání webových stránek přes síťové rozhraní. Toto rozhodnutí nerozdělit vykreslování a stahování do více modulů bylo přijato proto, že moderní webové aplikace již dávno nejsou statické dokumenty a síťová komunikace probíhá i poté, co je stránka načtena. Na úrovni modulu mohou být různé činnosti zajišťovány různými submoduly, ale na úrovni obecné architektury tyto prvky chápu jako jeden celek. Další částí aplikace je Integrační modul. To je část aplikace, která definuje API, a slouží k připojení na ostatní prvky softwarové základny, které chtějí využívat služby této aplikace. K tomuto modulu může být připojen buď uživatel skrze frontend, nebo jiná aplikace využívající funkcionality aplikace této. Jak bylo řečeno v kapitole 4.1.1, vývojem frontendu aplikace se nezabývám, a z důvodu toho, že jedním z cílů práce je ověřit návrh aplikace pomocí prototypu, nedeklaruji podrobné API pro komunikaci, protože se může v budoucnu změnit. Integrační modul není z technologického hlediska nijak zajímavý, ale je stěžejní pro použitelnost a snadnou integraci celé aplikace.

4.5 Možné přístupy k řešení

Možných přístupů k řešení se otevírá celá řada, proto je v této kapitole vyjmenuji a objasním jejich výhody a nevýhody. Nemá jít o ucelenou množinu všech myslitelných řešení a jejich zevrubnou analýzu, nýbrž o množinu způsobů, které se po prozkoumání oblasti jeví jako užitečné pro tuto práci. V dalších kapitolách využívám informace z kapitoly této a rozpracovávám řešení v podrobné analýze.

Základním kamenem této práce je vytvořit systém, pomocí kterého je možné zjistit, jak se na konkrétním zařízení a prohlížeči vykreslí zadaná webová aplikace. Komponenta, která má vykreslování na starosti, se obecně nazývá vykreslovací jádro. V rámci architektury na Obr. 1 Architektura návrhu (Zdroj: autor) je vykreslovací jádro součástí vykreslovacího modulu.

4.5.1 Implementace vlastního vykreslovacího jádra

Cestou, jak dosáhnout stanoveného výsledku, je implementace vlastního vykreslovacího jádra. Jednou z mála výhod je možnost napsat si kód podle svých potřeb. Nevýhodou je především přílišná náročnost z důvodu nutnosti implementovat chování všech reálných zařízení, ke kterým zřídkakdy existují specifikace. Dalším problémem je špatná udržovatelnost souladu se specifikacemi a standardy, kdy při každé změně je potřeba provést změnu v kódu. Cestou vývoje nového jádra se vydává málo projektů, ve většině případů, kde je potřeba vykreslovat webový obsah tvůrci využívají již existujících jader.

4.5.2 Využití reálných zařízení

Druhou cestou, kterou se vydává například projekt WebPageTest, je využití reálných zařízení, na kterých jsou spuštěny instance prohlížečů. Ty pak komunikují s řídící aplikací, která výsledky servíruje uživateli (WebPagetest comunity, 2014). Zásadní výhodou tohoto přístupu je přesnost a autentičnost výstupu, protože výsledek, přesně odpovídá výsledku reálnému, jsou totiž jeden a tentýž. Nevýhoda, která zamezuje použití tohoto řešení, popírá základní smysl této práce, a to odstranit nutnost fyzicky vlastnit všechna zařízení, jejichž výstup potřebuje uživatel znát. Vedle využití reálných zařízení se nabízí možnost využít emulátoru a všechna zařízení mít virtuálně na jednom stroji. Tím se odstraní nutnost vlastnit všechna zařízení a zjednoduší se správa infrastruktury. Stále to ale bude náročné z hlediska kapacity serverů, protože každá emulovaná instance potřebuje velké množství operační a diskové paměti. Správa i v tomto případě bude náročná, protože bude potřeba řídit tolik instancí, kolik zařízení budeme emulovat. Nepředpokládám využití těchto řešení, protože pro předpokládané využití převažují negativa. Na druhou stranu ve větším měřítku by to začalo dávat smysl. Dnes například již existuje služba pro testování nativních mobilních aplikací, kde je zapojeno několik stovek reálných zařízení (Xamarin Inc, 2017). Firma ovšem nepředpokládá takovou uživatelskou základnu, aby se provoz podobného řešení vyplatil.

4.5.3 Úprava existujícího vykreslovacího jádra

Na rozdíl od předchozích přístupů, se již pohybuji v oblasti, kde lze reálně uvažovat o rozpracování některého z návrhů. Jednou možnou variantou uchopení problému je upravení již existujícího vykreslovacího jádra k mému účelu. Výhodou tohoto řešení je přístup ke stabilnímu odladěnému kódu a zároveň volnost upravit ty části, které nevyhovují. Na rozdíl od implementace vlastního řešení od nuly není nutné psát veškerý kód vlastními silami. Nevýhodami jsou nutnost zorientovat se v cizím kódu a zároveň fakt, že mnoho kódu může zůstat nevyužito a zbytečně tak zabírat místo, znepřehledňovat kód a snižovat efektivitu celého řešení. Nevýhodou úpravy vykreslovacího jádra oproti použití existujícího, je nutnost koordinovat a začleňovat kód z nových vydání do vlastní verze v

případě, že chci získat jeho nové vlastnosti. V případě využití existujícího jádra bez jeho úpravy je omezením výsledného softwaru množina podporovaných vlastností v tom daném jádře.

Gecko

Vykreslovacích jader s volnou licencí je na trhu několik. V prohlížeči Mozilla Firefox je použito vykreslovací jádro Gecko (Mozilla Developer Network, 2014), které je použitelné nejen pro vykreslování HTML, CSS a Javascriptu, ale také XUL prvků, což je technologie pro tvorbu uživatelského rozhraní v XML. Vykreslovací jádro Gecko, používají, kromě webového prohlížeče Firefox, také ostatní produkty značky Mozilla: emailový klient Thunderbird, operační systém FirefoxOS (v době psaní této práce byl vývoj zastaven) a komplexní balík SeaMonkey. Dále je využito v dalších prohlížečích, většinou s nízkým tržním podílem, a například v kalendáři Sunbird a správci fotografií Google Picassa. Gecko má relativně dobrou podporu moderních webových technologií (Mozilla Developer Network, 2014) a v poslední době, kdy se součástí vykreslovacího jádra stal výkonný procesor javascriptu SpiderMonkey, je prohlížeč Firefox s jádrem Gecko i svižný. Kromě javascriptového procesoru jsou důležitými částmi Gecka parser dokumentů, systém řízení rozvržení (anglicky layout system), knihovny pro obrázky a multimédia, fonty, síť a další komponenty.

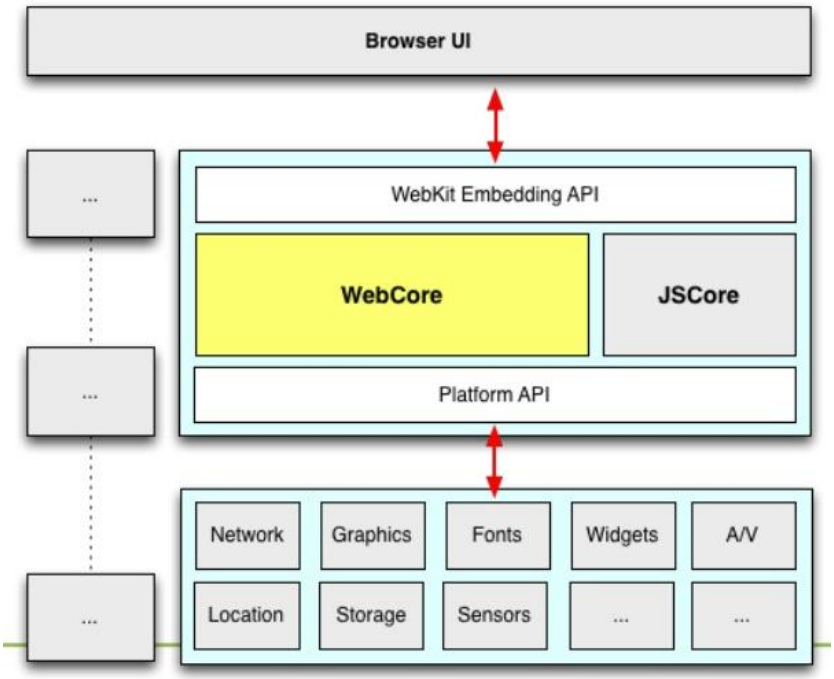
KHTML

Přibližně stejně starým projektem je jádro KHTML, to bylo použito v (nejen webovém) prohlížeči Konqueror, který je základním kamenem linuxového desktopového prostředí KDE. KDE je napsané pomocí grafické knihovny Qt, která je programátory široce využívaná a mimo jiné přinesla právě podporu zobrazování webového obsahu v desktopových aplikacích. Samotné KHTML v dnešní době z významnějších prohlížečů využívá pouze Konqueror.

WebKit

Projekt KHTML je z historického hlediska velmi důležitý, protože se stal základem donedávna nejrozšířenějšího vykreslovacího jádra WebKit. Společnost Apple pro svůj prohlížeč Safari potřebovala nové jádro a nejvhodnější se zdálo právě KHTML, a tak Apple na jeho základě vytvořil vlastní projekt, který později vydal pod volnou licencí.

WebKit není monolitickou aplikací, ale jen vrstva spojující několik komponent dohromady. Tyto komponenty jsou podobné jako v případě Gecka, ale na rozdíl od nich jsou použitelné samostatně, a je možné je vyměnit nebo odstranit. To je příklad vývojářských knihoven, které WebKit zpřístupňují vývojářům především desktopových aplikací nebo webového prohlížeče Google Chrome. Na Obr. 2 jsou znázorněny komponenty, ze kterých se Webkit skládá a s kterými komunikuje.



Obr. 2 Architektura vykreslovacího jádra Apple WebKit (Zdroj: QUANG, 2015)

Samotný WebKit je prostřední obdélník, který komunikuje s prohlížečovým rozhraním (horní obdélník), pomocí začleňovacího API, který slouží ke komunikaci se zobrazovacími komponentami. Díky tomu může být WebKit nezávislý na zobrazovací technologii a starat se pouze o vykreslování. Na druhé straně komunikuje jádro promocí takzvaného Platform API s komponentami, které poskytují data potřebná k vykreslení. Nejdůležitější komponentou v oblasti webu je síťová komponenta, která se stará o dodání dokumentů k vykreslení, neméně důležité jsou komponenty zajišťující přístup k systémovým písmům, úložným zařízením a v neposlední řadě také senzorům. Samotná vykreslovací komponenta se v původním WebKitu od společnosti Apple jmenuje WebCore a javascriptové jádro JSCore. WebCore parsuje HTML, CSS a ostatní markup dokumenty a staví z nich vykreslovací strom, který posléze vykresluje. Spolupracuje přitom s javascriptovým jádrem, který během vykonávání javascriptového kódu vykreslovací strom mění.

Vykreslovací strom lze připodobnit k dokumentovému objektovému modelu, protože hlavně DOMem je vykreslovací strom ovlivňován, ale v žádném případě nejsou totožné.

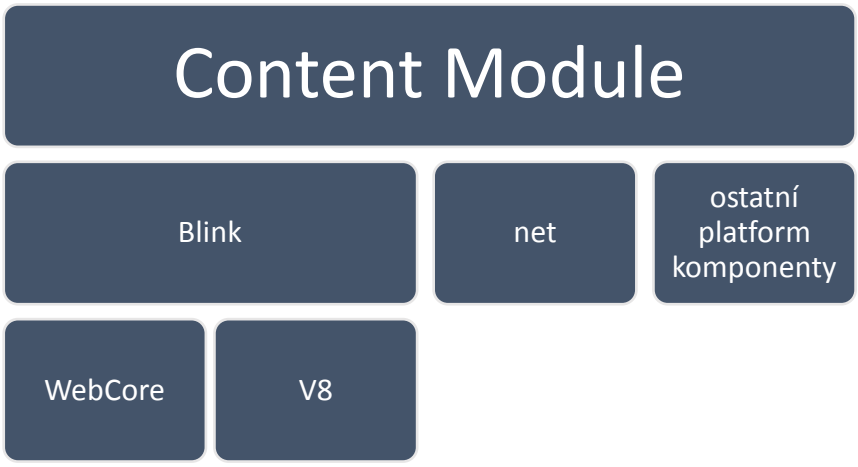
Jelikož má WebKit modulární architekturu, je možno jednotlivé moduly vyměnit za jiné. Například prohlížeč Google Chrome, který donedávna používal jako své vykreslovací jádro právě WebKit, nepoužíval JSCore, ale nahradil ho výkonnějším jádrem nazvaným V8 (ORFANO, 2011).

Jádro WebKit je velmi kvalitní, co se týká podpory webových standardů a rychlosti začleňování nových technologií, což je důvod, proč jej mnoho vývojářů upřednostňuje před ostatními jádry.

Blink

Prohlížeč Google Chrome je podle statistik (StatCounter, 2017) dominantním webovým prohlížečem. Ovšem vývoj Chromu probíhá převážně v projektu Chromium, což je webový prohlížeč oproštěný od integrovaných služeb Googlu. WebKit byl do nedávna vykreslovacím jádrem používaným v prohlížeči Google Chrome a Chromium, avšak Google se rozhodl vzít vývoj plně do svých rukou a založil vlastní projekt s názvem Blink. Blink je vykreslovací jádro, podobně jako WebKit, jenže na rozdíl od něj Blink nemůže běžet samostatně. Pro použití Blinku ve vlastním projektu je třeba využít komponenty Chromium Content Module, a to buď přímo nebo s využitím projektu Chromium Embedded Framework, který je určen pro portování Chromia a tudíž i jádra Blink do jiných aplikací.

Chromium Embedded Framework, zkráceně CEF, je knihovna napsaná v jazyce C++, která má usnadňovat integraci Chromia do jiných aplikací odstíněním vývojářů od



Obr. 3 Blink a příbuzné komponenty (Zdroj: autor)

interního fungování poskytnutím stabilního API. Projekt CEF je blíže rozebrán v kapitole 5.1.

Na Obr. 3 Blink a příbuzné komponenty (Zdroj: autor) je znázorněna struktura komponent pro integraci vykreslovacího jádra Blink tak, jak jich je využito ve webovém prohlížeči Chromium. Blink v Chromiu, stejně jako Apple WebKit, používá renderovací komponentu WebCore, ale zde je použito javascriptové jádro V8 místo původního JSCoru. Pro plnohodnotné využití funkcionality webového prohlížeče vedle Blinku využívá Chromium komponenty *net* a ostatních, takzvaných platform komponent, které, stejně jako u WebKitu, slouží k dodávání zdrojových dat pro vykreslení. Všechny tyto komponenty jsou zastřešovány Content Modulem, jehož jediným účelem je integrovat moduly na nižší úrovni, abstrahovat od jejich implementačních detailů a poskytnout vývojářům stabilní API pro komunikaci s tímto softwarem. Při pohledu na schémata architektur WebKitu a Blinku je patrné, že mezi oběma projekty existuje souvislost. Zajímavostí je, že na mnoha místech ve zdrojovém kódu Chromia je odkazováno na WebKit, ačkoli Blink je využíván už déle než dva roky.

4.5.4 Využití kompletní proxy

Dalším z možných přístupů k řešení je vytvoření proxy serveru, který by existujícímu prohlížeči dodával stránky v takové podobě, která pokrývá pouze určitou množinu podporovaných vlastností. Jinými slovy by aplikace odfiltrovala či přepsala ty části zdrojového kódu webové stránky, které prohlížeč nepodporuje.

Výhodou tohoto řešení je relativní jednoduchost implementace: jde pouze o vytvoření překladače zdrojových kódů stránky a vytvoření párů s údaji, na co se mají konkrétní části kódu přeložit. Tento přístup je použitelný v případě CSS souborů, protože ty jsou výrazně jednodušší než například javascriptové soubory, které obsahují mnoho proměnných.

Nevýhodou tohoto přístupu je jeho strnulost o obtížná modifikovatelnost u složitých transformačních pravidel a navíc fakt, že se pracuje ve skutečnosti s jinou webovou stránkou než byla aplikaci zadána. Ideálně by veškeré úpravy vlastností měly probíhat přímo ve vykreslovacím jádře. To by mělo za následek jednak snazší údržbu kódu a také jednodušší vhled do procesu vykreslení.

Jak jsem již popsal výše, považuji za vhodnější využití jiných prostředků pro dosažení výsledků.

4.5.5 Využití existující funkcionality webových prohlížečů

Během sbírání zdrojů pro tuto práci jsem zkoumal možné přístupy k interakci s běžící instancí prohlížeče, které je možné využít pro ovlivnění chování zobrazované webové stránky.

Zabudované ladící nástroje

Weboví vývojáři mají možnost využít v prohlížečích zabudovaných ladících nástrojů skrze které je možné měnit vzhled stránky za běhu. Mezi ně patří, mimo jiné, javascriptová konzole, informace o využití zdrojů (využití sítě, paměti, úložiště cookies, ostatních lokálních úložišť, aj.) a interaktivní prohlížeč dokumentového objektového modelu webové stránky. V prohlížeči Chromium je využíván takzvaný Chrome Debugging Protocol, což je standard pro programovou komunikaci s ladícími nástroji Chromia, využitelný z mnoha programovacích jazyků. (Google, 2017)

Funkcionalita je dostupná po spuštění Chromia se spouštěcím přepínačem --remote-debugging-port=XXXX, kde XXXX je číslo TCP portu, na kterém ladící rozhraní komunikuje. Komunikace využívá webových standardů WebSockets, http a JSON a tak kterýkoli programovací jazyk, který je podporuje, může být pro komunikaci využit. To ovšem není případ Pythonu 2.7, kde je podpora WebSockets omezená, a tak knihovna existuje pouze pro verzi 3 a vyšší. Pro javascript existuje projekt chrome-remote-interface dostupný přes balíčkovací systém NPM využívající prostředí Node.js. Tuto technologii jsem použil v první fázi testování přístupů k řešení a dospěl jsem k mnoha zajímavým poznatkům.

Chrome-remote-interface je snadno použitelná a jednoduchá knihovna pro komunikaci s Chromiem. Umožňuje dělat velkou část operací, kterou běžně uživatel prohlížeče provádí, včetně otevírání a zavírání karet, čtení jejich obsahu a práce s ladící konzolí. Díky programátorské přívětivosti knihovny jsem první verzi prototypu aplikace postavil právě na této technologii a ověřil jsem si, že díky ní bude snadné pořídit dva hlavní požadované výstupy z aplikace: snímek obrazovky a výsledný DOM stránky.

Prvotním omezením, na které jsem narazil, bylo, že je potřeba mít prohlížeč spuštěný, a to v tak velkém okně, v jaké velikosti jej potřebuji. To znamená, že pro testování televizorů je třeba spustit Chromium v okně o velikosti zobrazovací polochy onoho televizoru. Lze sice spustit Chromium se specifickou velikostí okna, ale při větších rozlišeních, než je dostupná velikost framebufferu spuštěného zobrazovacího serveru, snímek vykreslené stránky obsahuje jen viditelnou část obrazovky. (Zobrazovací server je v tomto významu komponenta operačního systému, která zobrazuje data z grafické karty na obrazovce uživatele. V GNU/Linuxových systémech je to většinou X server. Framebuffer je paměť obsahující obrazová data.). V případě, že by toto omezení bylo jediné, možným řešením pro použití na serveru připadá v úvahu spuštění emulačního X Serveru, například X Virtual Framebuffer Server. Takovýto X server má stejné aplikační rozhraní jako běžný X server, ale na rozdíl od něj nic nevykresluje na obrazovku, proto slouží hlavně ke spouštění grafických aplikací, kde není nutné a zároveň možné získat vizuální část aplikace.

Druhým a závažnějším problémem tohoto přístupu bylo, že umožňuje programově přistupovat jenom k funkcionalitě, která je dostupná v běžném Chromiu. Ovlivňování vzhledu webové stránky tak bylo možné, ale ve většinou až po samotném vykreslení

stránky. Takovéto ovlivnění již vykreslené stránky by v případě javascriptových API a CSS vlastností by poskytlo nerelevantní pohled na výsledek, protože různé výpočty rozvržení v průběhu vykreslování využívaly data, která v tu chvíli byla dostupná, avšak tento postup by je změnil až poté. Ideálním řešením je dané schopnosti prohlížeče vypnout, což Chrome Debugging Protocol neumožňuje.

Využití řádkových spouštěcích přepínačů

Další možností, jak využít stávající možnosti prohlížečů, je využití spouštěcích přepínačů vykreslovacího jádra Blink (The Chromium authors, 2014 [a]). Jimi je možné při startu prohlížeče ovlivnit spouštěnou funkcionalitu vykreslovacího jádra a tím vykreslování webové stránky. Problémem je, že při spouštění z příkazové řádky je možné pouze povolit či zakázat všechny, nebo jen některé funkce označené jako experimentální. Nelze tímto způsobem vypnout stabilní funkcionalitu prohlížeče. Dle dokumentace vykreslovacího jádra Blink jsou CSS vlastnosti uložené v jednom zdrojovém souboru, kde je možné je označovat za vypnutelné, popřípadě jim přiřadit vlastní obslužné metody. To se ovšem týká případu vývoje vlastního odštěpení (forku) Chromia nebo jádra Blink. Ačkoli je jádro a celý prohlížeč naprogramován v jazycích C a C++, se kterými jsem dosud nepracoval, využití tohoto přístupu se jeví jako nejvhodnější.

4.5.6 Shrnutí

Ve výsledku jsem nevybral jedno řešení, ale jejich kombinaci. V první řadě jsem vyloučil přístup využití reálných zařízení a jejich integrace do jednoho celku, protože toto řešení popírá jeden z předpokladů této práce, a toť nutnost vlastnit tato zařízení. Neubral jsem se cestou implementace vlastního jádra z důvodu příliš složité implementace a poměrně malých přínosů, což by bylo řešení neefektivní a neudržovatelné. Pro mé účely se, navzdory jednoduchosti implementace, nehodí ani Využití kompletní proxy, protože tato technika je velmi omezená, nepřesná a také těžko udržovatelná, ačkoli méně než tvorba vlastního jádra. Uvažoval jsem o jejím využití pro zpracování CSS souborů, ale to se mi povedlo vyřešit mnohem elegantněji. Vydal jsem se cestou využití existující funkcionality prohlížečů, možnosti při startu prohlížeče ovlivnit podporované funkce, a zároveň úpravy existujícího vykreslovacího jádra. Toto řešení považuji za vhodné z následujících důvodů. Využití standardní funkcionality je jednoduché na implementaci a dobře udržovatelné do budoucna. Úprava vykreslovacího jádra je sice náročná, ale přináší volnost v úpravách. Jak složité bude následné udržování souladu s hlavní vývojovou linií vykreslovacího jádra záleží pak už jen na rozsahu těchto úprav.

5 Použité technologie během implementace

5.1 Chromium Embeded Framework

Během zjišťování informací ohledně tématu jsem zjistil, že prohlížeč Google Chrome má nejen nejlepší podporu moderních technologií (DEVERIA, 2012-2017), ale také velice přívětivou infrastrukturu pro zabudování Chromia do vlastního projektu a navíc volně přístupný zdrojový kód, což otevírá možnost použít jej jako základ mé aplikace. Vývoj prohlížeče Google Chrome probíhá v open-source projektu s názvem Chromium, což je prohlížeč téměř stejný jako Google Chrome, jen není propojen se službami Googlu. Samotný prohlížeč se skládá z několika komponent, z nichž nejdůležitější je vykreslovací jádro. To v případě Chromia nese název Blink a stará se o převod zdrojových dat webové aplikace do vizuální podoby. Mimo něj v prohlížeči existuje také síťová vrstva, javascriptový procesor a zobrazovací komponenta.

Jak již bylo řečeno v jedné z předchozích kapitol, vykreslovací jádro Blink není navrženo tak, aby mohlo běžet samostatně. Jádro zprostředkovává svou funkcionalitu komponentě s názvem Content Module, která zastřešuje veškeré činnosti výše zmíněných komponent, kromě zobrazování, a poskytuje k nim jednotné API.

Samotné Chromium není monolitická aplikace. Kromě toho, že se skládá z různých modulů, běží v několika paměťově izolovaných procesech, které spolu komunikují pomocí zpráv IPC (Inter-process Communication), které jsou povětšinou asynchronní (The Chromium authors, 2014 [b]). Díky tomu je Chromium stabilnější a výkonnější, na druhou stranu to ale činí způsob vyzvedávání hodnot napříč procesy obtížnější.

Vývojář Marshall Greenblatt a komunita vyvíjí od roku 2008 open-source projekt s názvem Chromium Embedded Framework, zkráceně CEF. Jde o kód samotného Chromia, několika knihoven a obalových tříd pro usnadnění práce s integrací do vlastního projektu. Projekt sleduje hlavní vývojovou linii prohlížeče Chromium a pokaždé když vyšla nová verze prohlížeče Chromium, našel jsem na webu CEFu aktuální zdrojové kódy i instalační balíčky. To svědčí o aktivitě a tím pádem i o perspektivitě tohoto projektu.

Pro získání funkční knihovny CEF má uživatel několik možností. První z nich je stažení automaticky sestavovaných binárních balíčků, které jsou dostupné pro operační systémy MS Windows, macOS, Ubuntu a Debian GNU/Linux. Tento způsob je nejvhodnější pro většinu uživatelů, kteří mají v plánu využívat CEF tak, jak je a neměnit jeho zdrojový kód. Druhým způsobem je manuální překlad zdrojových kódů. To je proces velice komplikovaný, zdlouhavý a náročný na výpočetní výkon počítače i znalosti programátora. Je potřeba mít v systému dostupné různé knihovny ve specifických verzích, znát infrastrukturu projektu a vědět, jakým způsobem se projekt sestavuje. Postup je sice

uveden na stránkách projektu, ale při výskytu chyby při překladu, což je častá situace, je nutné mít více znalostí, než kolik poskytuje stručný návod. Výhodou tohoto postupu je možnost upravit zdrojové kódy dle potřeby a to jak obalových tříd Embedded Frameworku, tak Chromia samotného. Existuje ještě třetí možnost jak získat funkční CEF, a tou je využití automatického sestavovacího skriptu. Ten projde celým procesem od instalace závislostí přes stažení zdrojových kódů až po sestavení projektu. Stále jde o zdlouhavý proces, ale oproti ručnímu sestavování je výrazně jednodušší. Nevýhodou je omezená volnost programátora, protože automatický sestavovací skript je navržen pro získání aktuální verze projektu, nikoli pro úpravu jeho zdrojových kódů.

Na tomto místě je vhodné zmínit, že Chromium Embedded Framework je napsán z velké části v jazycích C a C++ a pro tyto jazyky také poskytuje rozhraní. Pro použití CEFu z jiných jazyků je proto vhodné použít další abstrakční vrstvu, která je popsána dále.

5.2 CEFPython

Projekt CEFPython je projekt vyvíjený Czarkem Tomczakem, který tvoří obalovou vrstvu Chromium Embedded Frameworku pro Python. Cílem projektu je zpřístupnit veškerou funkcionalitu Chromium Embedded Frameworku vývojářům v jazyce Python a tím jim umožnit využívat technologie prohlížeče Chromium. (Tomczak, 2016)

Součástí projektu jsou třídy v Pythonu, které se mapují na třídy CEFu, a další obslužný kód, který volá přímo C++ kód. To má několik konsekvencí. Zaprvé, tím, že nejde o čisté objekty Pythonu, nejde v mnoha případech použít inspekce objektů. Za druhé, ladění takovýchto programů je obtížné, protože místo obvyklého Pythonového výpisu výjimky (anglicky stacktrace nebo traceback) se často objeví jen hláška „segmentation fault“, jejíž význam je pokus o přístup do paměti, která není pro program alokována. Tento typ chyby není ve světě Pythonu obvyklý, protože běžně se v Pythonu operuje na vyšší úrovni a o přístup k paměti se stará virtuální stroj Pythonu. Výhodou komunikace s nativním kódem je jednak vyšší výkon aplikace, ale co je důležitější, zjednodušený vývoj tohoto projektu, oproti implementaci všech rozhraní v čistém Pythonu.

5.2.1 API CEFPythonu

API CEFPythonu je, dle mého názoru, přehledné a snaží se odrážet API samotného CEFu. Hlavními objekty jsou Browser a Frame. Instance třídy Browser, je běžící prohlížeč získaný voláním metody CreateBrowserSync v modulu cefpython. Je to hlavní třída celého projektu, přes kterou se lze dostat k dalším objektům CEFPythonu. Druhou důležitou třídou je Frame, která představuje jeden pohled do prohlížeče. Zjednodušeně lze říct, že Frame představuje načtenou webovou stránku, to ale pouze za předpokladu, že stránka

neobsahuje HTML značku <frameset> ani <iframe>. V tomto případě by webová stránka vytvořila tolik instancí třídy Frame, kolik rámců je v HTML dokumentu specifikováno. Objekty třídy Frame mají metody, které představují základní interakce uživatele se stránkou, jako například načtení URL nebo získání textu stránky, ale i pokročilejší uživatelské interakce, jako zobrazení zdrojového kódu (zobrazení dokumentového objektového modelu DOM není implementováno (Tomczak, 2017)), vyvolání javascriptové funkce, nebo nastavení hodnoty javascriptovému objektu window.

Objekty nastavení

Dalším typem objektů, které CEFPython vystavuje, jsou objekty nastavení. Ty jsou v současné době tři a jedná se o jednoduché obalové třídy, které přijímají pythonovský datový typ slovník (implementace obecné datové struktury hashmap) a jeho hodnoty doplňují o výchozí hodnoty a ty pak předávají do CEFu. Tyto třídy jsou si dost podobné a v některých případech jimi lze předat stejné nastavení. Hlavní rozdíl je v sémantice a v místě, kde se nastavení předává. Toto oddělení na tři místa vede ke snazší správě výchozích hodnot. První třídou týkající se nastavení je třída ApplicationSettings. Té se předávají hodnoty při inicializaci CEFPythonu a jsou určeny k jeho nastavení. Konfigurační hodnoty zde poskytnuté slouží jako výchozí hodnoty později vytvořeným prohlížečům. Prohlížeče, korektně instance třídy Browser, přijímají při svém vytváření argumenty v podobě zbylých dvou tříd. Myšlenka je podobná jako v originálním Chromiu. BrowserSettings představuje trvale nastavené hodnoty, které se projevují při každém spuštění prohlížeče. Jednotlivá spuštění prohlížeče může uživatel ovlivnit předáním argumentu typu CommandLineSwitches, které představují přepínače předávané Chromiu přes příkazovou řádku. Těchto přepínačů jsou desítky a ovlivňují chování prohlížeče v mnoha oblastech. Kompletní automaticky generovaný seznam je vystaven na webové stránce vývojáře Chromia Petera Beverloo (Beverloo, 2010).

Rozhraní

Použití CEFPythonu ve vlastním projektu předpokládá, že vývojář chce pozměnit chování některých částí prohlížeče. Architektura Chromia samotného je pro to uzpůsobená. Na místech, kde je programátorovi dána možnost vyřešit konkrétní akci vlastním kódem, je volána callback metoda. To je metoda předávaná jako argument jiné metodě, která je definována v konkrétním rozhraní. Programovací jazyk Python používá formu dynamického typování zvanou duck typing, což je způsob typování na základě implementovaných metod a nikoli deklarovaných rozhraní. Proto jsou v CEFPythonu rozhraní implementována jako třídy, jejichž metody jsou volány v rámci Chromium Embedded Frameworku na příslušných místech. Bohužel v CEFPythonu nejsou podporována všechna rozhraní, které je možné použít v CEFu. Seznam v současnosti podporovaných rozhraní ukazuje Tabulka 15.

Tabulka 15 Rozhraní implementovaná v CEFPythonu (Zdroj: TOMCZAK, 2016 [c])

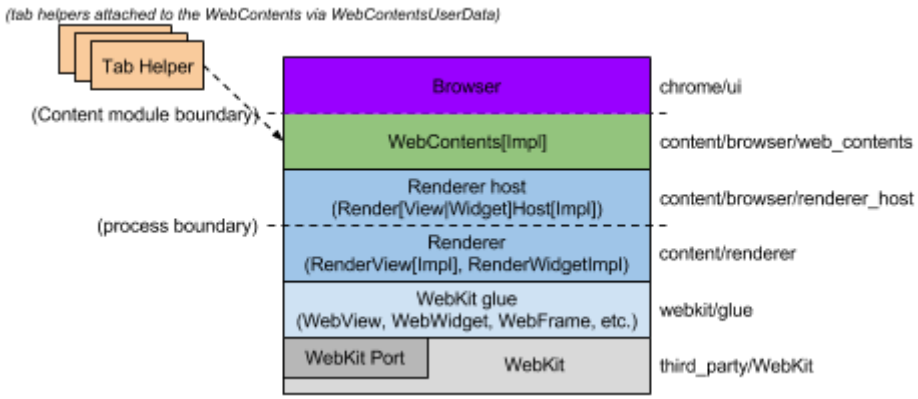
Název rozhraní	Účel rozhraní
DisplayHandler	Zpracovává události, které se týkají změny stavu zobrazení prohlížeče.
DownloadHandler	Spravuje stahování souborů a interakci s uživatelem. V současnosti není zcela implementováno.
FocusHandler	Implementuje tři metody, které jsou zavolány, když komponenta prohlížeče požaduje, získává nebo ztrácí fokus (zaostření na komponentu).
JavascriptDialogHandler	Umožňuje vytvořit vlastní implementaci vyskakovacích oken.
KeyboardHandler	Spravuje ovládání klávesnicí. Události jsou zavolány pouze pro hlavní frame.
LifespanHandler	Spravuje události týkající se životního cyklu prohlížeče.
LoadHandler	Spravuje události volané při iniciaci, změně stavu, ukončení a chybě načítání. Zatím neumí notifikovat o načtení DOMu (TOMCZAK, 2015).
RenderHandler	Umožňuje implementovat vlastní vykreslování komponent prohlížeče. Dostupné pouze v off-screen módu.
RequestHander	Spravuje události síťových požadavků prohlížeče.
ResourceHandler	Umožňuje implementovat vlastní síťové požadavky.
V8ContextHandler	Účelem je globální ošetření javascriptových výjimek. Zatím není plně implementováno.

Provázání s javascriptem

Důležitou funkcionalitou, kterou CEFPython disponuje je takzvaný javascript binding. Javascript binding je provázání objektů Pythonu a jejich vystavení v cílové webové aplikaci v javascriptovém objektu window. To umožňuje naprogramovat část aplikace v Pythonu a z javascriptu tyto vlastnosti využívat. Tímto způsobem lze implementovat různá API, která nejsou v Chromiu implementována. Toto je nejvhodnější a nejpřímočařejší postup, jakým rozšiřovat funkcionalitu javascriptu prohlížeče. Je nutné zdůraznit, že tímto způsobem se pouze vystavují objekty na úrovni platform komponent (vizte Obr. 3 Blink a příbuzné komponenty (Zdroj: autor) na straně 35), nejde o úpravu javascriptového jádra. Omezeními tohoto postupu jsou asynchronní komunikace mezi oběma jazyky, možnost provázat pouze určité datové typy a omezená možnost odchytávání výjimek, protože záleží na kontextu, ve kterém je výjimka vyhozena. Poslední zmíněný fakt pak ovlivňuje to, ve kterém prostředí by měla být výjimka odchytávána, zda v javascriptu, nebo v Pythonu.

Asynchronní komunikace je důsledkem toho, že Chromium Embedded Framework v aktuální verzi 3 využívá, v souladu s prohlížečem Chromium, architekturu několika procesů a meziprocesové komunikace mezi nimi, která je asynchronní (The Chromium authors, 2014 [b]). Javascript je vykonáván v kontextu vykreslovacího procesu, anglicky renderer process, kdežto kód v Pythonu je vykonáván v kontextu procesu prohlížeče, anglicky browser proces (Tomczak, 2016 [a]).

To, jakým způsobem jsou uspořádány procesy v prohlížeči Chromium, ukazuje Obr. 4



Obr. 4 Procesy v Chromiu (Zdroj: WILLSON, 2012)

Ačkoli je v obrázku odkazováno na starší vykreslovací jádro WebKit, které již není používáno, stále je tato architektura procesů platná, a je v souladu s pojmenováním komponent ve zdrojových kódech, kde se Blink stále nachází v adresáři *third_party/WebKit*. Je zde patrná hranice procesů (čárkovaně) mezi komponentami Renderer a Renderer host. Komponenty nad touto hranicí běží v procesu prohlížeče, komponenty pod ní ve vykreslovacím procesu. Dále je v obrázku naznačena linie, kde končí Content modul, ten v sobě zahrnuje všechny zde uvedené komponenty kromě uživatelského rozhraní, zde pojmenovaného Browser.

6 Postup Implementace

6.1 Získání snímku obrazovky

Vytváření snímků webových stránek bylo implementováno pomocí třídy handleru (vizte kap. 5.2.1 API CEFPythonu). Té jsem implementoval metodu OnPaint, která mezi argumenty přijímá, mimo jiné, i velikost aktuálně zobrazované plochy a surová data představující samotné pixely v běžném případě vykreslované na obrazovku. Tato data předávám knihovně PIL, která je převede do obrazového formátu PNG, a ten následně uložím. Pro použití ve formě webové aplikace namísto ukládání jsem uvážil dvě varianty. První je PNG obrázek kódovat do base64 řetězce a ten následně předávat uživateli. Druhá varianta je při přijetí korektně nastavené HTTP hlavičky Accept vrátit skutečný obrázek. Tyto dvě varianty mohou být v případě potřeby využívány obě, a rozhodnutí, jakou formou předat obrázek je ponecháno na klientovi.

6.2 Získání dokumentového objektového modelu webové stránky

Implementace získání dokumentového objektového modelu webové stránky není, bohužel, v CEFPythonu přímočarý proces. Jak vyplývá ze zmínek v sekci 5.2.1, konkrétně v tabulce s rozhraními implementovanými v CEFPythonu, veškerá funkcionalita týkající se DOMu je v CEFPythonu problematická. Buď není implementována vůbec, nebo je od ní naopak ustupováno z důvodu problematické implementace (TOMCZAK, 2015). Implementaci požadované funkcionality se snažím co nejvíce stavět na stabilních částech API CEFPythonu z důvodů lepší udržovatelnosti kódu do budoucna, ale pro čtení DOMu musím využít jiných cest. CEF sice má třídu CefDOMVisitor (GREENBLATT, 2016), jež je předávána třídě Frame, ale v CEFPythonu implementována není. V současnosti se této funkci nejvíce blíží GetSource, která ale vrací zdrojový kód, nikoli zpracovaný DOM, což je pro využití k analýze nedostatečné.

Jednou z možných cest, kterými bylo možné se vydat, bylo využití javascriptu. Pomocí jednoduchého volání lze získat DOM aktuální stránky:

```
window.document.getRootNode().documentElement.outerHTML
```

Problémem však zůstalo, jak výsledek dostat do aplikace a z ní jej předat uživateli, protože zde je překážkou meziprocesová asynchronní komunikace. Třída Frame má dvě metody pro spuštění javascriptu. První z nich, ExecuteFunction, přijímá textový řetězec, v němž je

název funkce k vykonání. To proběhne asynchronně, což znamená, že nevrátí výsledek ihned, a navíc zde není možné předávat funkci v Pythonu. Druhou metodou je ExecuteJavascript, která je podobná, jen vykonává jakýkoli javascriptový kód, ne pouze jedinou funkci, a přijímá i další parametry. Třetí způsob využití javascriptu je pomocí objektu JavascriptBindings, který je blíže popsán na straně 42 v sekci Provázání s javascriptem.

Nakonec jsem využil techniku provázání s Javascriptem. Metodou ExecuteJavascript volám objekt, který jsem vystavil pomocí JavascriptBindings. Tomuto objektu předávám pythonovský objekt StringIO. Ten následně plním DOMem a ten vracím klientovi. Tímto způsobem se mi podařilo obejít asynchronnost, protože předávání objektu napříč procesy využívá referencí, nikoli předávání hodnot, a tak je tento objekt dostupný napříč procesy.

Druhou možností interakce s javascriptem bylo využít Remote Debugging Protocol tak, jak je popsán v kapitole 4.5.5 Využití existující funkcionality webových prohlížečů. Této možnosti jsem se vyhnul, protože nepovažuji za vhodné přidávat do mého softwaru další vrstvu. Sice by některé postupy zjednodušila, ale nevyřešila by nejzávažnější stávající problémy, kterými jsou asynchronní komunikace a nutnost použití jiného programovacího jazyka.

6.3 Ovlivnění vykreslení

6.3.1 Kaskádové styly

Během studia řádkových přepínačů, anglicky command-line switches, jsem objevil přepínače --enable-blink-features a --disable-blink-features, které slouží k vypínání a zapínání experimentální funkcionality jádra Blink. Vyzkoušel jsem, že jejich použití je možné a skutečně ovlivňuje, jak jsou CSS styly aplikovány a to jak v Chromiu samotném, tak v CEFPythonu. Bohužel pro mne, experimentálních funkcí je v každém vydání Chromia jen poskrovnu, protože testování probíhá rychle a v dalším vydání je obvykle funkcionalita zařazena jako stabilní. To znamená, že mnohé CSS vlastnosti, jejichž podporu potřebuji umět dynamicky vypínat a zapínat, tímto způsobem ovlivnit nelze, ovšem pouze v běžných verzích. Logickým východiskem proto bylo sestavit si vlastní verzi Chromia, protože jsem neobjevil jiné dynamické ovlivnění jádra Blink v souvislosti s kaskádovými styly.

Proniknutí do zdrojových kódů Chromia není jednoduché, protože obsahují na třicet tisíc souborů, které jsou vzájemně provázané. Navíc neexistuje jednotná dokumentace popisující aktuální zdrojový kód. Existuje mnoho návodů na webu vývojářů Chromia, ale ty bývají zastaralé a neodráží současný stav projektu. Neaktuálnosti jsou i ve zdrojových

souborech samotných. Dobrým ilustrativním příkladem je přítomnost adresáře blink, ve kterém, navzdory očekávání, nejsou zdrojové kódy vykreslovacího jádra, ale skripty pro testy rozvržení vykreslené stránky. Na Blink samotný je mnohdy, ale ne vždy, odkazováno jménem původního vykreslovacího jádra WebKit. Zdrojové kódy Blinku se nacházejí v adresáři

```
src/third_party/WebKit/Source
```

navzdory tomu, že se nyní jmenuje jinak a není již produktem třetí strany. Níže v adresářové struktuře ve zdrojovém kódu v podadresáři

```
src/third_party/WebKit/Source/core/css/CSSProperties.in
```

se nachází seznam všech podporovaných CSS vlastností. Vlastnostmi se zde rozumí část specifikace CSS, která byla vývojáři Chromia implementována jako celek, nikoli pouze to, co je ve specifikaci CSS (The World Wide Web Consortium [W3C], 2011) nazýváno „property“. Příkladem vlastnosti je jednotka *rem*, property *background-size*, podpora *advanced selectors* nebo skupina více vlastností. Tento seznam slouží k vygenerování obslužného kódu vlastností v jádře Blink a nastavení specifického chování prohlížeče k těmto vlastnostem. Jednou z možností je vlastnosti přidělit její runtime příznak.

Runtime příznak je název položky v seznamu povolené funkcionality jádra Blink, který slouží k nastavení toho, jaké argumenty přijímají řádkové přepínače --enable-blink-features a --disable-blink-features. Tento seznam je uveden v souboru

```
src/third_party/WebKit/Source/platform/RuntimeEnabledFeatures.in
```

navzdory tomu, že dokumentace tvrdí, že by se mělo jednat o soubor ve formátu JSON s jiným názvem (The Chromium authors, 2014 [a]).Každá funkcionalita v tomto seznamu může mít přiřazen status, říkající, kdy je daná funkcionalita podporována. Možné stavy jsou *stabilní*, *experimentální*, *testovací* a *nepovolený*. Význam těchto stavů pro danou funkcionalitu MojeFunkcionalita je uveden v tabulce Tabulka 16 Stavy přepínačů funkcionality Blinku.

Tabulka 16 Stavy přepínačů funkcionality Blinku (Zdroj: The Chromium authors, 2014 [a])

Příkaz	Stav	Význam
MojeFunkcionalita status=stable	Stabilní	Tato funkcionalita je zapnuta.
Neuvedeno	Stabilní	Tato funkcionalita je zapnuta a nemá přiřazen runtime příznak.
MojeFunkcionalita status=experimental	Experimentální	Tato funkcionalita je ve výchozím stavu vypnutá, ale může být zapnuta pomocí přepínačů.
MojeFunkcionalita status=test	Testovací	Tato funkcionalita je dostupná pouze pokud je Blink spuštěn v režimu testů rozvržení. Není dostupná v Chromiu.
MojeFunkcionalita	Nepovolený	Funkcionalita není zapnuta.

Zajímavý je stav experimentální, který umožňuje funkcionalitu zapínat a vypínat podle potřeby. Pro ovlivnění kaskádových stylů, například podpory background-size, je nutné v souboru CSSProperties.in upravit řádek

```
background-size interpolable, custom_all
```

do podoby

```
background-size interpolable, custom_all, runtime_flag=CSSBackgroundSize
```

a v souboru RuntimeEnabledFeatures.in přidat řádek

```
CSSBackgroundSize status=experimental
```

Výše popsaný postup slouží k manuálnímu sestavení ze zdrojových kódů. Pro využití automatického sestavovacího skriptu je třeba použít upravený postup, který je popsán v kapitole 6.4.1.

6.3.2 Javascriptová API

Pro ovlivnění javascriptu je v aplikaci použito provázání javascriptu s Pythonem, jak je popsáno na straně 42. Tímto způsobem jsem byl schopen ovlivnit objekty dostupné z objektu window a tím pádem velkou část javascriptové funkcionality, protože většina testovaných vlastností a javascriptových API je dostupná právě tímto způsobem. Je třeba zmínit, že objekty dostupné z window je možné volat buď *window.nazev* nebo přímo *nazev*. To s sebou nese další obtíže, popsané dále. Mezi vlastnosti takto dostupné patří například localStorage nebo XmlHttpRequest. Co takto vypnutelné není, jsou objekty, které nejsou přímo dostupné z window objektu. CEFPython totiž zatím neumí provázat

javascript s Pythonem obousměrně. Je možné touto funkcionalitou vystavit pythonovský objekt ven do prohlížeče, ale nejde z Pythonu ovlivňovat vnitřní fungovaní čistých javascriptových objektů, ani nativního kódu. Příkladem, kde je tento postup prozatím nevyužitelný, je javascriptová funkce querySelectorAll, kterážto je funkcí objektu typu Node. Funkce querySelectorAll se používá pro výběr uzlů dokumentového objektového modelu stránky (pokud je voláno na kořenovém uzlu document) nebo podstromu některého uzlu (pokud je zavoláno na uzlu) který odpovídá zadanému CSS selektoru.

Využitím provázání s javascriptem jsem byl schopen do javascriptu vložit vlastní pythonovské objekty, a jelikož je mou snahou emulovat nepodporu technologie, tímto způsobem vkládám na místo původního objektu pythonovský objekt None, který se mapuje na javascriptové null. Efektem ve výsledné webové aplikaci je, že tento objekt je přepsán a není možné jeho funkcionality využít. Problém ale nastává u aplikací, které nějakým způsobem testují podporu této funkcionality, protože pokud je volán v javascriptu objekt, který neexistuje, je vrácena konstanta undefined, ale protože CEFPython mapuje None na null, může dojít k probémům, jak naznačuje programový výpis níže. V něm je ukázáno jak javascript vyhodnocuje různé výrazy. Konstanta *undefined*, stejně jako *null* je po přetypování na pravdivostní hodnotu nepravdivá. Z toho plyne, že porovnání hodnot těchto konstant vrací kladnou pravdivostní hodnotu. Ovšem typ obou konstant je rozdílný, proto navzdory stejné pravdivostní hodnotě striktní porovnávání operátorem === vrací zápornou pravdivostní hodnotu.

```
➔ Boolean(undefined)
false
➔ Boolean(null)
false
➔ Boolean(undefined == null)
true
➔ Boolean(undefined === null)
false
```

Fungování javascriptu v prohlížečích má ještě jednu konsekvenci. Jelikož objekt dostupný z objektu window není nutné volat přes operátor pro získání vlastnosti, tečkou, ale je také vystaven v nejvyšším javascriptovém kontextu, může docházet k následujícímu:

```
➔ neexistujícíObjekt
Uncaught ReferenceError: neexistujícíObjekt is not defined
    at <anonymous>:1:1
(anonymous) @ VM2388:1
➔ window.neexistujícíObjekt
undefined
```

Jelikož volání objektu přímo je volání přes referenci, která neexistuje, je z tohoto volání vystřelena výjimka *ReferenceError*. Naopak při volání objektu přes window, tedy

operátorem pro získání vlastnosti, se nehledá reference, nýbrž klíč v objektu window. Jelikož ten existuje, je vráceno undefined.

Výše zmíněné vlastnosti mohou ovlivnit, jakým způsobem je stránka vykreslena v případě, že vývojáři webové aplikace testují přítomnost javascriptového objektu jednou z metod, která rozlišuje konstanty *undefined* a *null*. Ve všech případech ale bude výsledek nepodpora funkcionality, lišit se bohužel bude reprezentace objektu.

6.4 Řešené problémy

6.4.1 Překlad ze zdrojových kódů

Kromě neaktuálnosti dokumentace a orientace ve zdrojových kódech Chromia byl největším problémem překlad projektu. Komplikací bylo, že jsem do této doby neměl zkušenosti s postupy překladu užívanými programátory v jazycích C a C++, ve kterých jsem nikdy dříve neprogramoval a proto mě to stálo mnoho úsilí. Celý postup probíhal v iteracích činností spustit překlad, přečíst chybovou hlášku, najít možné řešení a aplikovat ho. Takto jsem iteroval sedmkrát, než se mi povedlo projekt přeložit. Překlad je náročný nejen na znalosti, ale také na výpočetní výkon a úložnou kapacitu stroje, na kterém je překládáno. Samotný sestavovací proces na mém počítači běžel v řádu hodin. Nejdelší z nich dokonce hodin patnáct.

Ačkoliv se mi podařilo projekt úspěšně sestavit, výsledek nebyl spustitelný. Rozhodl jsem proto nejdříve vyzkoušet automatický sestavovací skript dodávaný s projektem CEFPython a sestavit projekt bez úprav. Tento skript spolupracuje se skriptem Chromium Embedded Frameworku a přidává kromě překladu také další možnosti, mezi nimiž je vytvoření pythonního instalačního balíčku. Pro rychlejší a pohodlnější práci jsem si nechal zřídit virtuální server, na kterém jsem sestavovací skript spouštěl. Sestavovací skript CEFPythonu byl v té době vyvinut pro překlad na GNU/Linuxové distribuci Ubuntu 12.04 nebo vyšší s běžným uživatelským účtem. Můj virtuální server, který odpovídá tomu, jak jsou virtuální servery ve firmě GrandIT běžně nakonfigurovány, je ale poháněn distribucí Debian ve verzi Jessie a pro práci na něm je používán účet root tak, jak je ve firmě zvykem. Kvůli tomu bylo nutné upravit sestavovací script, aby nepoužíval příkazu sudo pro získání administrátorského přístupu, a též bylo třeba doinstalovat balíky, které se v Debianu liší názvem od názvů používaných v distribuci Ubuntu. Tento postup se ukázal jako úspěšný a jeho výsledkem byla spustitelná instance projektu CEFPython.

Při studiu automatického skriptu jsem objevil, že projekt CEFPython při sestavování knihoven CEFu aplikuje vlastní rozdílové soubory do nadřazeného projektu, a tímto způsobem ovlivňuje jeho zdrojové soubory těsně před překladem. Následoval jsem tento

postup a vytvořil jsem rozdílový soubor obsahující změny popsané v kapitole Ovlivnění vykreslení v sekci 6.3.1. Tento rozdílový soubor byl následně uložen v adresáři patch ve zdrojových souborech CEFPythonu a byl přidán odpovídající záznam do konfiguračního souboru patch.py umístěného tamtéž. Konfigurační soubor přijímá dvě hodnoty: název rozdílového souboru bez přípony a cestu v adresáři zdrojových kódů CEFu, odkud bude rozdíl aplikován. Tento způsob úpravy zdrojových kódů a překladu je velmi pohodlný a nenáročný z pohledu údržby. Má však jednu nevýhodu, která souvisí s charakterem mého softwaru a rychlostí jakou je Chromium vyvíjeno. Soubor RuntimeEnabledFeatures.in se mění v každém vydání Chromia, ve kterém byla přidána funkcionalita vykreslovacího jádra, což má za následek nutnost upravit rozdílový soubor, protože ten se stává neaplikovatelný, pokud nenajde části zadané ve zdrojovém souboru. Složitost spočívá v tom, že při využití automatického skriptu je stahování, konfigurace a překlad zdrojových souborů součástí jednoho volání metody a tudíž nejde do tohoto procesu jednoduše vstoupit. Z tohoto důvodu je vhodné nejdříve prohlédnout zdrojové kódy, např. na internetu, upravit rozdílový soubor a následně spustit skript. Je důležité sledovat výpis skriptu, protože při chybě aplikování rozdílů není skript ukončen, ale je pouze vypsána chybová hláška a skript běží dál.

Doporučený postup překladu zdrojových kódů CEFPythonu, CEFu a Chromia, je uveden v dokumentaci projektu CEFPython (TOMCZAK, 2017 [a]).

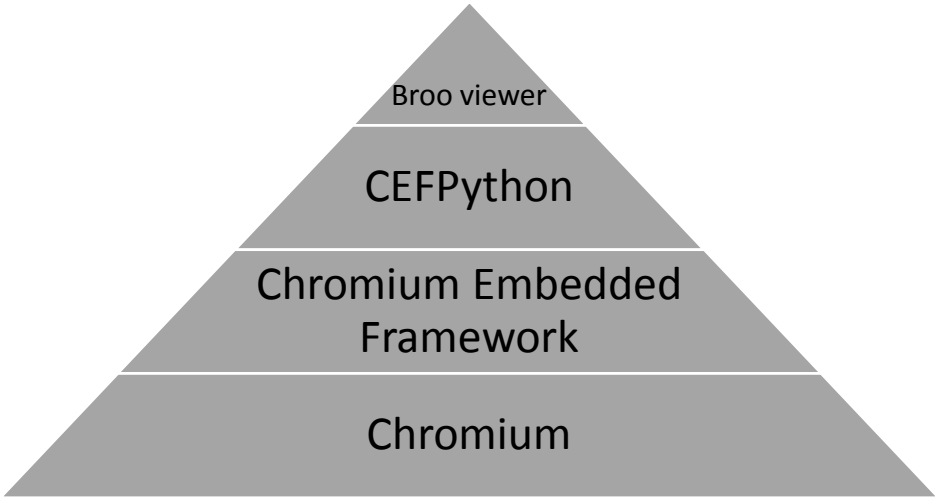
6.4.2 Spouštění instance v prostředí bez zobrazovacího serveru

Na zprovoznění možnosti spouštět Chromium v prostředí bez zobrazovacího serveru, takzvaně v headless módu, měli vývojáři plán od roku 2015, nicméně funkcionalita byla ve stabilním Chromiu povolena až s verzí 56 (The Chromium Authors, 2016), která vyšla 26. ledna 2017, nicméně do CEFPythonu se dostala až 1. března 2017 (TOMCZAK, 2017 [b]), tudíž nebylo v mých silách prozkoumat tento způsob spouštění a implementovat aplikaci s jeho využitím. Místo toho jsem implementoval spouštění na serveru způsobem, jaký se používal doposud, a to s využitím Xvfb X Serveru. Tato technologie je představena v kapitole 4.5.5.

6.4.3 Rychlost vývoje použitých projektů

Na jednu stranu je příjemné pracovat s aktuální verzí softwarového produktu a vědět, že již probíhá práce na dalších vylepšených verzích, nicméně je náročné udržovat vlastní projekt v soudržnosti s takto rychle vyvíjenými projekty. Zvláště v situaci, kdy projekty na sebe přímo navazují. Na Obr. 5 Návaznost projektů (Zdroj: autor) je zobrazeno, jakým způsobem. Na vrcholu pyramidy je software zpracovávaný jako součást této práce, pracovně nazvaný Broo viewer. Ten je implementován pomocí CEFPythonu, který

zpřístupňuje funkcionalitu Chromium Embedded Frameworku. A konečně, ten využívá zdrojových kódů webového prohlížeče Chromium.



Obr. 5 Návaznost projektů (Zdroj: autor)

7 Testování

Testování proběhlo dle testovacích scénářů v kapitole 4.3.3. Níže v tabulkách jsou uvedena vstupní a výstupní data testů a zhodnocení. Testování probíhalo dne 9.4.2017. Testy jsou rozděleny do tabulek dle kategorií testovacích scénářů, podobně, jako je tomu v plánu testů. Jednotlivé testy mají svůj identifikátor, odkaz na scénář, podle kterého bylo testováno, nastavení v jakém software byl v průběhu testu a dále vstupní a výstupní data a výsledek testu. Vstupní a výstupní data se liší dle charakteru testů napříč tabulkami.

Hodnota ve sloupci Nastavení softwaru nabývá dvou hodnot: testovací a produkční. Testovací nastavení znamená, že softwaru byla přímo zapnuta či vypnuta konkrétní funkcionalita a ostatní zůstaly povoleny. Tohoto stavu bylo dosaženo úpravou modulu pro získání vlastností tak, aby vlastnosti nezískával z Broo, ale byly mu nastaveny ručně. Tento stav byl zaveden proto, aby bylo možné ověřit vlastnosti nezávisle na ostatních vlastnostech či podmínkám. Produkční nastavení se odkazuje na stav, ve kterém softwaru není nic přednastaveno ručně a veškeré jeho činnosti probíhají automatizovaně, včetně získávání dat o schopnostech požadovaného prohlížeče.

Jak je uvedeno v kapitole 4.3.1 věnující se testovacímu prostředí, testy vlastností proběhly oproti webovým stránkách vystavených na lokálním serveru. Zdrojový kód těchto stránek je uveden v přílohách a je na ně odkazováno z tabulek testů ve sloupcích s názvem *Vstupní data* u testů 1 a 8, resp. *Vstupní data stránka* u testů 9 a 11. V těchto stránkách jsem použil výhradně vlastních zdrojových kódů, včetně obrázku *border.png* pro testy 3, 10 a 11. Obrázek *background.png* pro testy 2, 10 a 11 je ve veřejné doméně (anglicky public domain), tudíž je volně k užití.

Výstupní obrázková data testů 1 a 7 byla pořízena ve velikosti 300 x 300 px, ale pro použití v tabulkách byla oříznuta pro zobrazení pouze relevantního obsahu, protože velkou část plochy zabíralo bílé místo. Podobný je případ výstupních obrázků u testu 10 a 11, kde výsledné obrázky mají správnou velikost jako zobrazovací plocha mobilního telefonu Nokia 5800d a Google Nexus 5, jak bylo požadováno. Pro lepší čitelnost i tyto obrázky byly oříznuty.

7.1 Komentář výsledků testů

Tabulka 17 Testování vykreslení (Zdroj: autor) přináší testování schopnosti softwaru ovlivnit vykreslovací funkcionalitu prohlížeče. V prvním testu je testována CSS vlastnost box-shadow. Jak je patrné z výstupních dat, se zapnutou podporou této vlastnosti je zobrazen pod žlutým obdélníkem černý stín, u vypnuté podpory stín chybí. Výsledkem testu je, že tuto vlastnost software umí ovlivnit.

Druhý test ověřuje vlastnost backround-size. Se zapnutou podporou je obrázek pastelek výrazně menší než u podpory vypnuté. Výsledek tohoto testu je kladný. Tento test není zaměřen na CSS vlastnost background-repeat, která ovlivňuje opakování obrázku pozadí. To může být vodorovné, svislé, žádné či obojí. Poslední zmíněné je výchozí, a proto je v tomto testu výsledek opakující se. I v případě, že by se obrázek neopakoval, byl by tento test vyhodnocen kladně, protože je zaměřen izolovaně na vlastnost background-size.

Testování podpory border-image je předmětem testu 3. Žlutému obdélníku je předán obrázek se vzorem, jenž má tvořit okraj obdélníku. Je vidět, že i podporu této vlastnosti umí software podmíněně vypínat.

Zajímavým testem je test č. 4. V něm je na HTML stránce žlutý obdélník s textem, přes nějž, taktéž s textem, zasahuje obdélník červený. Ten má navíc CSS vlastnost opacity (česky neprůsvitnost) nastavenou na polovinu. Lze si všimnout, že průhledné není jen červené pozadí objektu, ale taktéž jeho text. Naopak v pokud prohlížeč tuto vlastnost nepodporuje, je horní obdélník celý červený a není skrze něj vidět obsah spodního obdélníka.

Podobně jako testování box-shadow, je i test 5. zaměřen na stínování, v tomto případě textu. Z pohledu na výsledky je zřejmé, že test dopadl korektně.

V 6. testu je podbarvený element s pevnou šířkou, v němž je vepsán text obsahující dlouhé slovo, jenž přesahuje šířku elementu. Standardní chování zakazuje lámání slova, pokud není nastavena hodnota break-word vlastnosti word-wrap. V prvním výsledku prohlížeč tuto hodnotu vlastnosti podporuje, a tak je slovo zalomeno na několika místech tak, aby nepřeteklo šířku elementu. Ve druhém případě prohlížeč tuto hodnotu nezpracovává, a tak text přeteče přes pravý okraj. Software se i zde chová korektně.

Sedmý test se věnuje testování podpory javascriptového lokálního úložiště s názvem localStorage. Ve stránce je skript, který do úložiště uloží hodnotu „localStorage funguje“ a tu opět vytáhne a vypíše ji do dokumentu. Následuje ještě jeden příkaz pro výpis, tentokrát statického textu, ten slouží pro ověření, zda prohlížeč pokračuje ve vykonávání skriptu. Pro ověření, že chyba v jednom skriptu neovlivní chod skriptů ostatních, je ve stránce ještě druhý skript, který taktéž vypisuje statický text. V případě absence podpory localStorage je možné vidět, že se software zachová správně. Nejen, že se nevypíše text ukládaný do úložiště, ale vypadne výjimka, která přeruší vykonávání skriptu. Nicméně následující skript není touto výjimkou ovlivněn, a tak se text „Pokračujeme dále“ korektně vypíše na obrazovku.

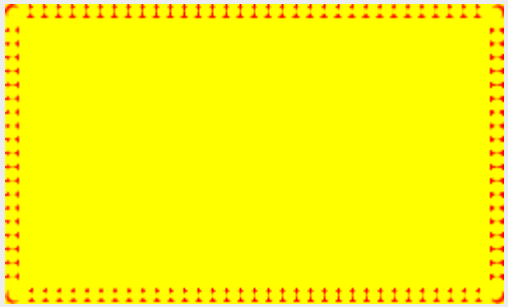
Stejnou funkcionalitu, ale jiný typ požadovaného výsledku ukazuje Tabulka 18. Její výsledky ukazují, že výpis dokumentového objektového modelu stránky koresponduje s vykreslením v testu 7. Kromě možnosti vypínat a zapínat podporu lokálního javascriptového úložiště je v tomto testu ověřeno, že je software schopen uživateli předat DOM stránky.

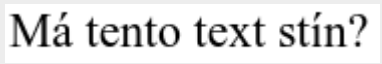
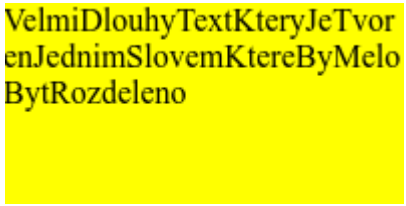
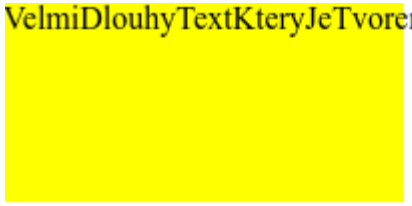
Tabulka 19 ukazuje výsledky testu, zda vrácená textová reprezentace dokumentového objektového modelu stránky není pouze jejím zdrojovým kódem. Test probíhá oproti stejné webové stránce, jako test 7 a 8, v němž se testuje podpora localStorage. DOM, na rozdíl od zdrojového kódu stránky, obsahuje změny provedené javascriptem, v tomto případě přidané elementy s textem. Jak je vidět, výsledky testu odpovídají popisu očekávaného stavu a tudíž je tento test úspěšný.

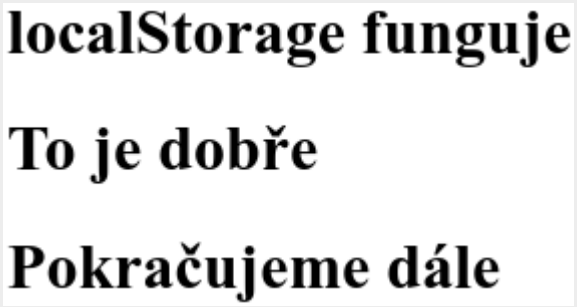
Testy 10 a 11, jež popisuje Tabulka 20, ukazují, že je software schopen komunikovat s rozpoznávačem Broo, stáhnout z něj data o schopnostech prohlížečů, a na základě těchto dat vykreslit komplexní stánku se zohledněním nepodporovaných vlastností. Vybrány byly mobilní telefenony, jejichž prohlížeče kvalitativně nacházejí na opačných stranách spektra. V testu 10 byla testována Nokia 5800d, která, jak lze vidět z výsledného vykreslení, nepodporuje lokální úložiště, průhlednost elementů, obrázkové ohraničení, škálování velikosti obrázku pozadí a stínování elementu div, avšak umožňuje využít stínování textu. V testu 11 byla stejná stránka zobrazena z pohledu mobilního telefonu Google Nexus 5, který, dle Broo, všechny její vlastnosti dokáže využít. Oba tyto testy úspěšně ověřily, že je software schopen ovlivnit vykreslení stránky dle vlastností prohlížeče, jehož user-agent byl softwaru předán.

Test 12, který ukazuje Tabulka 21 Testování alternativního scénáře, ověřuje, že prototyp softwaru disponuje alespoň základní úrovní stability, tj. nezhavaruje v případě neexistence požadované webové stránky a zachová se dle scénáře.

Tabulka 17 Testování vykreslení (Zdroj: autor)

Id testu	Id scéná ře	Nastaven í sw	Vstupní data	Výstupní data obrázek Zapnutá podpora	Výstupní data obrázek Vypnutá podpora	Výsledek testu
1	TC1	testovací	Příloha C:			OK
2	TC2	testovací	Příloha A:			OK
3	TC3	testovací	Příloha B:			OK

Id testu	Id scéná ře	Nastaven í sw	Vstupní data	Výstupní data obrázek Zapnutá podpora	Výstupní data obrázek Vypnutá podpora	Výsledek testu
4	TC4	testovací	Příloha D:			OK
5	TC5	testovací	Příloha E:			OK
6	TC6	testovací	Příloha F:			OK

Id testu	Id scéná ře	Nastaven í sw	Vstupní data	Výstupní data obrázek Zapnutá podpora	Výstupní data obrázek Vypnutá podpora	Výsledek testu
7	TJS1	testovací	Příloha G:			OK

Tabulka 18 Testování DOMu (Zdroj: autor)

Id	Id scénáře	Nastavení sw	Vstupní data	Výstupní data DOM stránky Zapnutá podpora	Výstupní data DOM stránky Vypnutá podpora	Stav testu
8	TJS1	Testovací	Příloha G:	Příloha I:	Příloha J:	OK

Tabulka 19 Ověření, zadali dom není zdrojovým kódem (Zdroj: autor)

Id	Id scénáře	Nastavení sw	Vstupní data stránka	Vstupní data user-agent	Výstupní data	Stav testu
9	TJS2	Produkční	Příloha G:	Mozilla/5.0 (Linux; Android 4.4; Nexus 5 Build/KRT16H) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.105 Mobile Safari/537.36	Příloha I:	OK

Tabulka 20 Testování komplexního vykreslení (Zdroj: autor)

Id	Id scénáře	Nastavení sw	Vstupní data stránka	Vstupní data user-agent	Výstupní data Obrázek	Stav testu
10	TI2	Produkční	Příloha H:	Mozilla/5.0 (SymbianOS/9.4; Series60/5.0 Nokia5800d-1/60.0.003; Profile/MIDP-2.1 Configuration/CLDC-1.1) AppleWebKit/533.4 (KHTML, like Gecko) NokiaBrowser/7.3.1.33 Mobile Safari/533.4		OK

Id	Id scénáře	Nastavení sw	Vstupní data stránka	Vstupní data user-agent	Výstupní data Obrázek	Stav testu
11	TI2	Produkční	Příloha H:	Mozilla/5.0 (Linux; Android 4.4; Nexus 5 Build/KRT16H) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.105 Mobile Safari/537.36		OK

Tabulka 21 Testování alternativního scénáře (Zdroj: autor)

Id	Id scénáře	Nastavení sw	Vstupní data stránka	Vstupní data user-agent	Výstupní data	Stav testu
12	TA1	Produkční	http://localhost/neexistujici.html	Mozilla/5.0 (Linux; Android 4.4; Nexus 5 Build/KRT16H) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/30.0.1599.105 Mobile Safari/537.36	Bílý obrázek o velikosti 1080 x 1920 px Prázdný DOM Software nezhavaroval	OK

8 Závěr

Tato práce se věnovala návrhu softwaru, který ukazuje, jak se webová stránka zobrazí v konkrétním prohlížeči na určitém hardwaru. Po zevrubné analýze byly vybrány technologie, které se pro použití v takovémto softwaru hodí, a byl pomocí nich vytvořen prototyp aplikace. Ten byl podroben testům, které potvrdily, že dává korektní výsledky. Tím bylo prokázáno, že návrh předložený touto prací je proveditelný.

Hlavním přínosem této práce je položení základního kamene pro testování kompatibility technologií užitých ve webových aplikacích s prohlížeči, skrze které je k těmto aplikacím přistupováno, nejen ve firmě GrandIT. Od této práce očekávám zjednodušení procesu testování napříč mobilními telefony a televizory, což je v současnosti ve firmě nejčastějším problémem.

8.1 Dosažení cílů

- Cíl navrhnout software, který ukáže vykreslení webové stránky v konkrétních prohlížečích a hardwaru byl splněn. Na základě analýzy byly vybrány nejvhodnější technologie pro využití v projektu a s jejich pomocí byl vytvořen prototyp softwaru. Ten, jak testy ukázaly, dosvědčuje, že návrh a vybrané technologie jsou pro toto použití vhodné.
- Vedlejší cíl dodat teoretické informace ohledně fungování webových technologií byl splněn v ještě větší míře, než jsem očekával. V průběhu analýzy a implementace jsem narazil na mnoho informací, které ovlivnily mé rozhodování a proto jsem tyto informace v práci zdůraznil. Mezi ně patří například architektura procesů a jejich asynchronní komunikace v prohlížeči Chromium, samotná architektura komponent prohlížeče, či stav jeho zdrojových kódů. Těchto informací může čtenář s podobnými cíli využít, protože v této práci se vyskytují pohromadě a jsou dostatečně okomentovány.
- Taktéž všechny další vedlejší cíle byly naplněny. Problém současného stavu byl dostatečně vymezen a byla popsána a zhodnocena relevantní současná řešení, která se snaží tento problém řešit. Dle návrhu, jak zlepšit podmínky vývoje a testování webových aplikací, jsem obhájil návrh vlastního softwaru, zanalyzoval jsem požadavky na něj a implementoval jsem prototyp. Ten jsem v poslední fázi práce úspěšně otestoval dle vytyčeného plánu testů.

8.2 Možná budoucí rozšíření

Odevzdáním práce pro mne práce na tomto projektu nekončí, protože je třeba kolem této aplikace vyvinout potřebnou softwarovou infrastrukturu a rozvinout ty části návrhu, které dostaly nižší prioritu, a proto nebyly v této práci realizovány. Nicméně i další potenciální zájemci mohou z této práce vycházet. Předpokládám, že jim nejvíce pomůže zde prezentovaný způsob, jak rozšiřovat, či upravovat prohlížeč Chromium. Na něm je postaveno mnoho projektů, a jak ukázala tato práce, platforma Chromia je pro vývoj tohoto typu projektů vhodná.

Existuje software pro automatizované testování webových stránek s názvem Selenium IDE, který využívá propojení na různé prohlížeče, mezi nimi i Google Chrome. Dovedu si představit, že s využitím poznatků z této práce bude možný autor schopen vytvořit produkt kombinující Selenium IDE a Chromium pro integrační testování webových aplikací z prohlížečů s různou úrovní podpory technologií.

Další možností, jak rozvést výstupy této práce, je využít můj software k vykreslování webových stránek na serveru. To sice v tomto projektu probíhá, ale dovedu si představit, že někdo tento koncept využije například k optimalizaci načítání webových aplikací účinnou komprimací na serveru.

Příloha A: Testovací stránka background-size

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 200px;
    height: 100px;
    background: url(background.jpg);
    background-size: 30px 30px;
}
</style>
</head>
<body>

<div></div>
<br>

</body>
</html>
```

Příloha B: Testovací stránka border-image

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 200px;
    height: 100px;
    background-color: yellow;
    padding: 15px;
    border: 10px solid transparent;
    border-image: url(border.png) 30 round;
}</style>
</head>
<body>

<div></div>
<br>

</body>
</html>
```

Příloha C: Testovací stránka box-shadow

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 200px;
    height: 100px;
    background-color: yellow;
    box-shadow: 30px 30px 1px #000;
}
</style>
</head>
<body>

<div>Hello</div>
<br>

</body>
</html>
```

Příloha D: Testovací stránka opacity

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    position: absolute;
    top: 0;
    left: 0;
}
.bottom {
    background-color: yellow;
    width: 200px;
    height: 100px;
}
.upper {
    background-color: red;
    opacity: 0.5;
    width: 100px;
    height: 200px;
}
</style>
</head>
<body>

<div class="bottom">div dole, div dole, div dole</div>
<div class="upper">div nahore<br /> div nahore</div>
<br>

</body>
</html>
```

Příloha E: Testovací stránka text-shadow

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    font-size: 1.5em;
    text-shadow: 5px 5px 2px black;
}
</style>
</head>
<body>

<div>Má tento text stín?</div>
<br>

</body>
</html>
```

Příloha F: Testovací stránka word-wrap

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
    width: 200px;
    height: 100px;
    background-color: yellow;
    word-wrap: break-word
}
</style>
</head>
<body>

<div>VelmiDlouhyTextKteryJeTvorenJednimSlovemKtereByMeloBytRozdeleno</div>
<br>

</body>
</html>
```

Příloha G: Testovací stránka localStorage

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  </head>
  <body>
    <script>
        window.localStorage.setItem("a", "localStorage funguje")
        document.write("<h1>" + window.localStorage.getItem("a") + "</h1>")
        document.write("<h1>To je dobře</h1>")
    </script>
    <script>
        document.write("<h1>Pokračujeme dále</h1>")
    </script>
  </body>
</html>
```

Příloha H: Testovací stránka sdružená

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
    background-color: #ccf;
}
div {
    position: absolute;
    top: 200px;
    left: 0;
    font-size: 1.5em;
    text-shadow: 5px 5px 0px black;
    word-wrap: break-word;
}
.bottom {
    width: 200px;
    height: 100px;
    background: url(background.jpg);
    background-size: 30px 30px;
    padding: 15px;
    border: 10px solid transparent;
    border-image: url(border.png) 30 round;
    box-shadow: 30px 30px 1px #000;
}
.upper {
    background-color: red;
    opacity: 0.8;
    width: 100px;
    height: 200px;
}
</style>
</head>
<body>
<div class="bottom">div dole, div dole, div dole</div>
<div class="upper">div nahoře<br /> div nahoře</div>
<script>
    window.localStorage.setItem("a", "localStorage funguje")
    document.write("<h1>" + window.localStorage.getItem("a") + "</h1>")
    document.write("<h1>To je dobře</h1>")
</script>
<script>
    document.write("<h1>Pokračujeme dále</h1>")
</script>
</body></html>
```

Příloha I: Výsledek testu při povolené localStorage

```
<html><head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <script>
        window.localStorage.setItem("a", "localStorage funguje")
        document.write("<h1>" + window.localStorage.getItem("a") + "</h1>")
        document.write("<h1>To je dobře</h1>")
    </script><h1>localStorage funguje</h1><h1>To je dobře</h1>
    <script>
        document.write("<h1>Pokračujeme dále</h1>")
    </script><h1>Pokračujeme dále</h1>



</body></html>
```

Příloha J: Výsledek testu při nepovolené localStorage

```
<html><head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <script>
        window.localStorage.setItem("a", "localStorage funguje")
        document.write("<h1>" + window.localStorage.getItem("a") + "</h1>")
        document.write("<h1>pokracujeme dale</h1>")
    </script>
    <script>
        document.write("<h1>Pokračujeme dále</h1>")
    </script><h1>Pokračujeme dále</h1>



</body></html>
```

Terminologický slovník

Termín	Význam
Vývojář	Programátor, kodér, designér, nebo vývojový tým podílející se na vzniku aplikace
API	Programové rozhranní softwaru
Framebuffer	*„Vyhrazená část paměti počítače nebo grafické jednotky, do které se ukládají údaje pro zobrazení jednotlivých hodů snímku na monitoru.“* (VIKTORIN, 2005)
DOM	*„Dokumentový objektový model webové stránky je (...) rozhraní, které umožňuje programům dynamicky měnit obsah strukturu a styl dokumentu.“* (The World Wide Web Consortium [W3C], 1997)
User agent	Řetězec, jímž se prohlížeč identifikuje v rámci http požadavku. Je předáván jako hlavička.
Http hlavička	Součást http požadavku, která samotný požadavek popisuje
Opacity	Neprůhlednost. Určuje míru, nakolik je zkrze element vidět. Udává se v intervalu [0;1], kde 0 znamená úplně průhledná (neviditelná) a 1 znamená neprůsvitná
Proxy	Služba stojící před jinou službou zprosředkující přídavné funkce. Např. vyvažování zátěže, nebo úpravu obsahu.
Spouštěcí přepínač	Argument předávaný programu při spuštění z příkazové řádky
Monolitická aplikace	Opak komponentové aplikace. Veškeré služby, které aplikace poskytuje jsou pevnou součástí aplikace. Služeb nelze využít samostatně.
Webová aplikace	Služba, k níž je očekáván přístup přes http/https protokol z webového prohlížeče, napsaná především pomocí technologií HTML CSS a javascript.
Webová stránka	Část webové aplikace dostupná uživateli po zadání specifického URL.

Použité informační zdroje

ADAPTIC, s.r.o., 2005. *Frontend. In Slovníček* [Online] [cit. 2.3.2017]
Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/frontend/

Afilias Technologies Ltd, 2016. *Properties. In Device Atlas* [Online] [cit. 2.2.2016]
Dostupné z: https://deviceatlas.com/resources/available-properties

ARTAIL, H. a RAYDAN, M., Device-Aware Desktop Web Page Transformation for Rendering on Handhelds. *Personal and Ubiquitous Computing*, 2005, **9**(6).

BALAYN, S., 2014. *caniuse-api.* [Online] [cit. 2.5.2016]
Dostupné z: https://www.npmjs.com/package/caniuse-api

BEVERLOO, P., 2010. *List of Chromium* XE "Chromium" *Command Line Switches.* [Online]
Dostupné z: http://peter.sh/experiments/chromium-command-line-switches/

BUCHALCEVOVÁ, Alena, *Metodiky budování informačních systémů.* Praha: Oeconomica, 2009. ISBN 978-80-245-1540-3

(DEVERIA, Alexis. 2012-2017). *Can I use repository*. [Online-databáze]
Dostupné z: https://github.com/Fyrd/caniuse/

FOWLER, Martin. 2014. *Microservices.* [Online][cit. 6.2.2017]
Dostupné z: https://martinfowler.com/articles/microservices.html XE "HTML"

(Free Software Foundation, Inc, 2007) GNU AGPL - *GNU Affero General Public License.* Free Software Foundation, 2007 XE "HTML"

FRYČ, Lukáš. *Visual Testing of Browser Screen Captures.* Brno, 2010. Bakalářská práce. Masarykova Univerzita. Fakulta Informatiky. Vedoucí práce Marek GRÁC.

Google, 2017. *Chrome Debugging Protocol.* [Online] [cit. 4.4.2017]
Dostupné z: https://developer.chrome.com/devtools/docs/debugger-protocol

GREENBLATT, Marshall. 2016. *Class CefDOMVisitor.* [Online]
Dostupné z:
http://magpcss.org/ceforum/apidocs3/projects/(default)/CefDOMVisitor.html XE "HTML"

HANYŠ, Pavel, *Využití HTML5 při vývoji webových aplikací* Praha, 2011. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky. Vedoucí práce Alena BUCHALCEVOVÁ

CHEN, Teh-Chung, DICK, Scott. a MILLER, James, Detecting visually similar Web pages: Application to phishing detection. *ACM Transactions on Internet Technology*, 2010, **10**(2).

JÍRŮ, Michaela. *Možnosti srovnávání obrázků v mobiních aplikacích.* Praha 2016. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky. Vedoucí práce Jarmila PAVLÍČKOVÁ

KOKOŠKA, Lukáš.*Srovnání Broo, interní dokument.* Čtyřkoly, 2013. Interní dokument.

KRČMÁRIK, Petr. *Záznam a přehrávání uživatelských akcích v internetových prohlížečích.* Brno, 2016. Diplomová práce. Masarykova univerzita. Fakulta informatiky. Vedoucí práce Tomáš REBOK

MIŠÁK, Josef. *Porovnání renderování internetových prohlížečů pro různé operační systémy.* Brno, 2013. Diplomová práce. Masarykova univerzita. Fakulta Informatiky. Vedoucí práce Marek GRÁC

Mozilla Developer Network, 2014. *Gecko. In Internals* [Online] [cit 11.12.2016] Dostupné z: https://developer.mozilla.org/en-US/docs/Gecko/FAQ

NADRCHAL, Tomáš, *Přístupy k řešení mobilních webových aplikací.* Praha, 2012. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky. Vedoucí práce Zuzana ŠEDIVÁ

(Open Source Initiative, 1999). *The 3-Clause BSD License.*

Open Source Initiative, 1999.
Dostupné z: https://opensource.org/licenses/BSD-3-Clause

ORFANO, Finn., 2011. *Google Chrome for Linux: Why V8 Bests WebKit* XE "WebKit" *'s JavaScript Engine.* [Online] [cit. 23.2.2017] Dostupné z: http://www.brighthub.com/computing/linux/articles/19626.aspx

POKLUDA, Václav, 2008. *Wapová proxy pro webový obsah.* Praha. Diplomová práce České vysoké učení technické v Praze. Fakulta elektrotechnická. Vedoucí práce Vlastimil Pečínka

QUANG, Nguyen., 2015. *Web browser architecture.* [Online] [cit. 14.11.2016] Dostupné z: https://image.slidesharecdn.com/web-browserarchitecture-150609231155-lva1-app6892/95/web-browser-architecture-1-638.jpg?cb=1433891674

ScientiaMobile, Inc, 2016 [a]. *How does WURFL work?.* [Online] [cit 6.4.2016] Dostupné z: http://wurfl.sourceforge.net/wurfl_schema.php

ScientiaMobile, Inc, 2016 [b]. *WURFL.* [Online] [cit 6.4.2016] Dostupné z: http://wurfl.sourceforge.net/

SKLENÁŘ, Pavel, *Testování webových aplikací za pomoci detekce změn jejího vzhledu.* Praha, 2012. Diplomová práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky. Vedoucí práce Luboš PAVLÍČEK

StatCounter, 2016. *Mobile and tablet internet usage exceeds desktop for first time worldwide.* [Online] [cit. 2.1.2017] Dostupné z: http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide

StatCounter, 2017. *Browser Market Share Worldwide.* [Online] [cit. 29.1.2017] Dostupné z: http://gs.statcounter.com/

The Chromium XE "Chromium" Authors, 2014 [b]. *Inter-process Communication.* [Online] [cit. 4.4.2017] Dostupné z: https://www.chromium.org/developers/design-documents/inter-process-communication

The Chromium XE "Chromium" Authors, 2014 [a]. *Runtime Enabled Features.* [Online] Dostupné z: https://www.chromium.org/blink/runtime-enabled-features

The Chromium XE "Chromium" Authors, 2016. *Headless Chromium.* [Online][cit. 26.3.2017]
Dostupné z: https://chromium.googlesource.com/chromium/src.git/+/56.0.2924.56/headless/

(The World Wide Web Consortium [W3C], 2011) *CSS* XE "CSS" - *Syntax and basic data types.* The World Wide Web Consortium 2011.
Dostupné z: https://www.w3.org/TR/CSS21/syndata.html XE "HTML" #rule-sets

(The World Wide Web Consortium [W3C], 2016) HTML XE "HTML" 5.1 *Specifikace značkovacího jazyka HTML.* The World Wide Web Consortium 2016.
Dostupné z: https://www.w3.org/TR/html51/

(The World Wide Web Consortium [W3C], 2007) DOM *Dokumentace dokumentového objektového modelu.* The World Wide Web Consortium 2007.
Dostupné z: https://www.w3.org/DOM/

TOMCZAK, Czarek. 2015. *Implement LoadHandler.OnDomReady.* [Online] [cit.28.1.2017]
Dostupné z: https://github.com/cztomczak/cefpython/issues/32

TOMCZAK, Czarek. 2016 [c]. *API* XE "API" *categories.* [Online] [cit.28.1.2017]
Dostupné z: https://github.com/cztomczak/cefpython/blob/cefpython55/api/API-categories.md

TOMCZAK, Czarek. 2016 [b]. *cefpython.* [Online] [cit.28.1.2017]
Dostupné z: https://github.com/cztomczak/cefpython

TOMCZAK, Czarek. 2016 [a]. *Javascript bindings.* [Online] [cit. 20.12.2016]
Dostupné z: https://github.com/cztomczak/cefpython/blob/master/api/JavascriptBindings.md

TOMCZAK, Czarek. 2017 [c]. *Architecture Features.* [Online] [cit.1.4.2017]
Dostupné z: https://github.com/cztomczak/cefpython/wiki/ArchitectureFeatures unor 2017

TOMCZAK, Czarek. 2017 [a]. *Build instructions.* [Online] [cit.2.2.2017]
Dostupné z: https://github.com/cztomczak/cefpython/blob/master/docs/Build-instructions.md

TOMCZAK, Czarek. 2017 [b]. *Update to Chromium* XE "Chromium" *v56 on Linux.* [Online] [cit.5.4.2017]
Dostupné z: https://github.com/cztomczak/cefpython/commit/28828651ee0cba6638989dca253ca56f29598afd

ROSSUM, Guido van, Warsaw, Barry & Coghlan, Nick, 2001. *PEP 8 Style Guide for Python* XE "Python" *Code.* [Online] [cit. 24.3.2017]
Dostupné z: https://www.python.org/dev/peps/pep-0008/

VIKTORIN, Jan. 2005. *Slovník pojmů počítačové grafiky.* [Online] [cit. 13.4.2017]
Dostupné z http://www.fa.vutbr.cz/pages/centra/vypocetni/vyuka/slovnik.htm

WARTIAK, Rastislav. *Performance Testing Tool for Web Applications.* Praha, 2007. Bakalářská práce. Univerzita Karlova v Praze. Fakulta Matematicko-fyzikální. Vedoucí práce Tomáš Kalibera.

WebPagetest Comunity, 2014. *Mobile Testing. In WebPagetst* [Online] [cit 2.4.2016] Dostupné z: https://sites.google.com/a/webpagetest.org/docs/system-design/mobile-testing

WILSON, Brett. 2012. *How Chromium* XE "Chromium" *Displays Web Pages.* [Online] [cit.13.4.2017] Dostupné z: https://www.chromium.org/developers/design-documents/displaying-a-web-page-in-chrome

XAMARIN Inc, 2017. *Devices.* [Online] [cit. 6.2.2017] Dostupné z: https://testcloud.xamarin.com/devices

Rejstřík

— A —

API, 11, 15, 17, 18, 20, 36, 39, 40, 41, 43, 44, 45, 47, 49, 52, 81

— B —

Blink, 4, 40, 41, 43, 44, 47, 48, 50, 51, 52

— C —

CEF, 40, 44, 45, 46, 49, 54, 55
CEFPython, 5, 45, 46, 47, 49, 50, 52, 53, 54, 55
CSS, 6, 16, 19, 28, 30, 31, 32, 38, 39, 41, 43, 50, 51, 53, 57, 58, 81

— D —

dokumentový objektový model, 14, 25, 27, 28, 29, 40, 42, 46, 49, 53, 58, 59

— H —

HTML, 6, 11, 16, 17, 19, 38, 39, 46, 58, 65, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 79, 81

— Ch —

Chromium, 5, 40, 41, 42, 44, 45, 46, 47, 48, 50, 55, 66, 67, 79, 81, 82

— J —

javascript, 6, 19, 42, 47, 53
Javascript, 38

— P —

prohlížeč, 11, 13, 14, 16, 19, 23, 38, 39, 40, 41, 42, 43, 44, 45, 48, 50, 51, 53, 55, 57, 58, 59, 66, 67
Python, 23, 42, 45, 46, 47, 50, 53, 81

— W —

WebKit, 4, 16, 39, 40, 41, 48, 51, 80