Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Webová aplikace pro organizaci sportovních závodů a akcí

Bc. Tomáš Litera

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Tomáš Litera

Informatika

Název práce

Webová aplikace pro organizaci sportovních závodů a akcí

Název anglicky

Web application for organizing sports competitions and events

---

Cíle práce

Cílem práce je připravit a uvést do provozu registrační informační systém určený pro organizování sportovních závodů a akcí. Zajistit účastníkům závodu rychlou a snadnou registraci a pořadatelům nástroj na zpracování zadaných dat a jejich využití před, v průběhu a po závodu.

Metodika

Tato práce navazuje na bakalářskou práci Informační systém pro zpracování závodů v reálném čase a dále ji rozvíjí. Práce se zabývá problematikou realizace webových aplikací v dnešním světě a požadavky, které jsou na ni kladeny. Práce analyzuje současnou situaci a požadavky na webové aplikace jako celek, dále je dělí na dílčí problémy a vlastnosti. Jednotlivé problémy a vlastnosti jsou dále podrobovány měřením a experimentům, a jsou porovnávána různá nalezená řešení. Nejlepší řešení jsou poté vybírána a implementována na zkušební aplikaci. Řízení vývoje a implementace probíhá za pomoci agilních metodik.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

web, aplikace, informační systém, scrum, mobil, desktop, tablet, cache, rychlost

---

Doporučené zdroje informací

Böhmer M., Návrhové vzory v PHP, Brno: Computer Press, 2012, ISBN 978-80-251-3338-5
Cederholm, D., Webdesign s webovými standardy, Brno: Zoner Press, 2004, ISBN 80-86815-15-3
Kadlec, V., Agilní programování – Metodiky efektivního vývoje softwaru, Brno: Computer Press, 2004, ISBN 80-251-0342-0
Ries, E., Lean Startup, Brno: BizBooks, 2015, ISBN 978-80-265-0389-7
Řezáč, J., Web ostrý jako břitva, Baroque Partners s.r.o., 2014, ISBN 978-80-87923-01-6
Sklar, D., PHP5 – moduly, rozšíření a akcelerátory, Brno: Zoner Press, 2005, ISBN 80-86815-19-6

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Mgr. Vladimír Očenášek, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 21. 10. 2016

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 10. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 08. 03. 2017

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Webová aplikace pro organizaci sportovních závodů a akcí“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne

Poděkování

Rád bych touto cestou poděkoval vedoucímu diplomové práce Mgr. Ing. Vladimíru Očenáškovi za konzultace, odbornou pomoc, korekturu, dobré nápady a trpělivost. Taktéž vyjadřuji své vřelé poděkovaní RNDr. Janu Fischerovi, PhD. za poskytování podkladů pro další práci, RNDr. Davidu Svobodovi, PhD., Ing. Janě Karaové za jazykovou korekturu práce a nakonec také své ženě Ditě a synovi Jonášovi za nekonečnou trpělivost při dokončování vysokoškolského studia.

Webová aplikace pro organizaci sportovních závodů a akcí

Web application for organizing sports competitions and events

Souhrn

Práce se zabývá problematikou tvorby informačních systémů, jež jsou určené pro pořádání závodů a sportovních akcí. Práce navazuje na bakalářskou práci Informační systém pro zpracování sportovních závodů v reálném čase a dále ji rozvíjí. Jsou popsány technologie a metodiky určené pro tvorbu WWW stránek se zaměřením na frontend aplikace. Tedy na design, uživatelskou přívětivost a jednoduchost, rychlost a na přizpůsobení se obrovskému množství existujících mobilních zařízení. Jsou též nastíněny metodiky, jak realizaci takového projektu plánovat a řídit, a jedna z agilních metodik je v implementační části práce i vyzkoušena.

Summary

The Thesis deals with the development of information systems, which are designed for organizing races and sports events. At the same time this Thesis builds on the bachelor thesis Information system for real-time processing of sports events and it expands it further. The technologies and methodologies for development of web pages with focus on the frontend application are described in this Thesis. Thus it focuses on its design, user friendliness and simplicity, speed and last but not least to adaptation to the great number of today existing mobile devices. There are outlined the methodologies of how to plan and control implementation of such a project. And one of the agile methodologies is used and tested.

Klíčová slova: Frontend, Webdesign, Cache, Mobilní zařízení, Responzivní design, Javascript, CSS, Sprites, Open Source, TDD, Testování, Agilní metodiky, Bootstrap, UX

Keywords: Frontend, Webdesign, Cache, Mobile devices, Responsive design, Javascript, CSS, Sprites, Open Source, TDD, Testing, Agile, Bootstrap, UX

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Masivní rozšíření internetu vyneslo do popředí zájmu webové aplikace. Jedním z druhů těchto aplikací jsou informační systémy (IS). Umožnují publikovat obsah na web, ukládat, sdílet a spravovat soubory, řídit společnosti a pomáhat při rozhodování, třídit, uchovávat a zpracovávat informace. Staly se tak nedílnou součástí našeho života a dennodenně je používáme, protože nám ulehčují práci.

Právě takový informační systém je předmětem této práce. Ale tentokrát ne z pohledu serveru a aplikačního pozadí (tzv. backendu), ale právě z pohledu koncového uživatele, zákazníka, člověka, který takový systém bude používat a využívat. Tedy právě pohled zepředu (tzn. frontend).

Právě pohled na aplikaci z pozice uživatele má svá největší úskalí. Zde už nejde jenom o to, aby aplikace fungovala a aby fungovala dobře. Zde jde hlavně a především o to, aby byla použitelná, aby se s ní dobře pracovalo. Aby uživateli svým rozhraním zpříjemňovala a zrychlovala práci a nekomplikovala ji. Aby byla použitelná na různých zařízeních od počítače, přes tablet až po mobil. Aby byla vzhledově příjemná. Zkrátka aby se dobře používala.

A to vše a ještě mnohem více je nutné, aby informační systém, neboli webová aplikace, uspěla v dnešním internetu a u dnešního uživatele.

Hlavní snahou textu a autora je podat relativně ucelený pohled na dnešní způsob návrhu a následný vývoj a ladění frontendové aplikace. Dále se práce také zabývá i tím, kdo za danými kroky stojí a jakým způsobem pracuje. V průběhu textu tak na příkladu bude objasněno kdo je to kodér nebo UX designer, co je wireframe nebo mobile-first přístup, dále pak také co je Scrum, Lean development a vůbec agilní programování.

Realizačním záměrem pak tedy není vytvořit samotnou aplikaci, nýbrž její uživatelské rozhraní. A to takovým způsobem, aby bylo uživatelsky přívětivé a použitelné na rozmanitém množství zařízení.

Cílem je vytvořit moderní multiplatformní aplikaci, která odpovídá anebo se alespoň blíží k dnešním standardům uživatelské přívětivosti aplikačního rozhraní.

2 Cíl práce a metodika

Cílem práce je navrhnout a realizovat podobu přístupného a přívětivého uživatelského rozhraní pro informační systém určený pro skautský vodácký závod Napříč Prahou – přes 3 jezy. Tato práce tak přímo navazuje na bakalářskou práci Informační systém pro zpracování závodů v reálném čase a dále ji rozšiřuje a rozvíjí. Vedlejším cílem je pak popsat podobu, postupy a metodiky návrhu a realizace dnešních uživatelských rozhraní webových aplikací.

Tato práce si neklade za cíl vytvořit plně funkční uživatelské rozhraní se všemi jeho funkcemi a vlastnostmi. Hlavním cílem je připravit alespoň základ takové aplikace, která bude moci být v budoucnu dále rozšiřována o požadované funkce a vlastnosti bez větších problémů. Zároveň si tato práce klade za cíl uvést a popsat souvislosti a procesy, které probíhají při realizaci a vývoji daného uživatelského rozhraní a u uživatelských rozhraní webových aplikací obecně.

Úvod práce je zaměřen na seznámení se s dnešní podobou webdesignu. Jak dnešní weby fungují, jak jsou strukturovány, jak vypadají procesy jejich návrhu a prototypování, kdo za takovou prací stojí a v neposlední řadě i jaké používá technologie. Dále pak práce rozebírá i samostatné procesy práce v týmu při návrhu webu - agilní metodiky. Scrum, Kanban, Lean development a další. V neposlední řadě také rolemi v dnešním webdesignu.

Vlastní práce se zabývá tvorbou návrhu uživatelského rozhraní, jeho skicováním a prototypováním, volbou designového frameworku a jeho použitím při výstavě rozhraní. Právě v této časti je pak řešen hlavně responzivní design a mobile-first návrh. V neposlední řadě se věnuje také samotné rychlosti uživatelského rozhraní (rychlost načítání, reakce na požadavek), a to hlavně kvůli použití na mobilních zařízeních.

Závěr práce pak zhodnocuje celkový průběh návrhu vývoje, snaží se shrnout chyby a nedostatky, které se během daných fází objevily.

3 Přehled řešené problematiky

V minulosti existovala komunita nadšenců, kteří v přítmí své garáže, rušeném pouze charakteristickým svitem monochromatického monitoru, vytvářeli systémové utility, databázové aplikace nebo celé účetní systémy.

Nelze jednoznačně říci, že tento přístup již patří minulosti, nicméně současný vývoj softwaru – především toho rozsáhlejšího – probíhá obvykle jinak. Software je obvykle vyvíjen týmově, musí být kvalitnější, musí být k dispozici rychleji a vyžaduje podporu a údržbu i v době dávno po svém nasazení. Všechny tyto aspekty výrazně mění celý proces vývoje a požadavky na něj. [8]

3.1 Softwarové inženýrství dnes

Firmy zabývající se vývojem programových produktů již před časem pochopily, že bez nějakého řádu, bez nějakého vymezení firemních postupů a bez stanovení určitých pravidel je vývoj rozsáhlejších systémů v lepším případě neefektivní, v tom horším pak zhola nemožný.

Není však účelné, aby si každá jednotlivá firma vymýšlela své vlastní postupy a pravidla: je to i zbytečné, protože v průběhu let byla vytvořena celá řada komerčních i volně dostupných produktů, které jsou pro firmy ihned k dispozici a které přinášejí odpovědi na naznačené otázky. Jinými slovy, existuje celá řada metodik vývoje softwaru.

Stejně jako se vyvíjí a mění svět kolem nás, mění se také software a požadavky na něj. A mění se také zákazníci – zadavatelé – a jejich potřeby. V důsledku toho se mění také nároky na vývojový tým, přeneseně pak i na celou společnost „vyrábějící“ software. Z těchto důvodů se logicky vyvíjejí a mění také metodiky – tak, aby vždy pokrývaly aktuální požadavky na software a na jeho tvorbu. [11, 6, 8]

3.1.1 Co je softwarové inženýrství

Jak již bylo řečeno v úvodu, pro pochopení metodik je důležité i základní povědomí o jakémsi obecném rámci, uvnitř kterého se veškeré diskuse odehrávají. Tímto rámcem je právě softwarové inženýrství, které si dále rozebereme.

Pro účely této práce byla zvolena definice, kterou přednesl Fritz Bauer, jeden z duchovních otců softwarového inženýrství. Tato definice poprvé zazněla na konferenci NATO v roce 1968, kde se pojem „softwarové inženýrství“ živě diskutoval.

„**Softwarové inženýrství je zavedení a používání řádných inženýrských principů tak, abychom dosáhli ekonomické tvorby softwaru, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.**“

V této krátké a výstižné definici je skryto všechno podstatné, co softwarové inženýrství obsahuje, čím se zabývá a o co se snaží.

Především: softwarové inženýrství nedbá jen na vlastní zavedení určitých inženýrských principů, ale pokouší se nabádat i k jejich dodržování. Mezi těmito dvěma pojmy je v praxi poměrně dramatický rozdíl: zavést lze cokoliv, dokonce se můžeme tvářit, že se to používá, nicméně pokud to ve své podstatě není pravda, výsledky nebudou dokonalé.

V definici se dále vyskytuje pojem **ekonomická tvorba softwaru**. Pokud tento pojem rozšíříme na „po všech stránkách úspěšnou“ tvorbu softwaru, můžeme najít řadu faktorů, které by měly být k naplnění tohoto cíle splněny. Příklady těchto faktorů jsou uvedeny v následujícím přehledu:

- **vhodně sestavit vývojový tým**
- **volba správného vývojového nástroje**
- **provést úvahu „vyvinout/koupit“**
- **nalézt společnou řeč se zadavatelem**
- **uvažovat o budoucí údržbě/rozšiřování**

Takových bodů bychom nalezli celou řadu a všechny můžeme zahrnout do pojmu „úspěšná a ekonomická tvorba sw“, protože zanedbání kterékoliv z nich povede k ekonomickým (i jiným) ztrátám – buď ihned, nebo v budoucnu.

Softwarové inženýrství je také souhrn mnoha aspektů. Definice dále hovoří o tom, že vytvořený software má být **spolehlivý** a má **účinně pracovat na dostupných technologických zařízeních**. O tom se asi netřeba dlouze rozepisovat: opět se dostáváme zpět k tomu, že nestačí napsat program, ale výsledná aplikace by měla splňovat požadavky uživatelů, nikoliv potenciálních *hackerů*. Navíc by měla efektivně využívat dostupné zdroje – ať již hardwarové, softwarové, personální či jiné. [6, 5, 14]

3.1.2 Metodiky, životní cykly a procesy vývoje softwaru

Metodologie, metodika, metoda – v souvislosti s těmito produkty softwarového inženýrství dochází velmi často ke zmatení pojmů a k nedorozumění. Vyskytují se totiž tři různé pojmy, které se často zaměňují.

Metodologie je neobecnější pojem a znamená ve své podstatě „nauku o metodikách“. Jinými slovy, pod pojmem metodologie se skrývá vědní disciplína, která nějakým způsobem rozebírá metodiky, definuje je apod.

Metodika je nejdůležitější pojem, s nímž se budeme setkávat. Skoro stejně lze používat také pojmy „model životního cyklu“ nebo „vývojový proces“. Pro účel této práce budeme považovat tyto pojmy za téměř synonymické. Budeme je používat k označení komplexních postupů a návodů pro vývoj softwarové aplikace. Pod metodikou se skrývají všechny etapy řešení (u vývoje softwarové aplikace tedy jde o všechny fáze životního cyklu); metodika se zabývá spíše pohledem z výšky, hledáním nadhledu, hledáním odpovědí na otázky proč?, kdo?, kdy? a co?

Metoda je pak označení pro nějaký konkrétní postup vedoucí k vyřešení dílčího problému. [6]

3.1.3 Agilní metodiky

O tom, že dnešní svět je dynamický a mění se prakticky na každém našem kroku, není třeba psát zdlouhavá pojednání. Stejně tak se zřejmě shodneme, že prakticky totéž platí pro vývoj softwaru: jedná se o jednu z nejdramatičtěji proměnlivých oblastí.

Především u internetových projektů, na něž se snažíme v této práci klást zvláštní důraz, je nejpalčivějším problémem rychlost vývoje. Vyvíjí-li společnost webovou aplikaci dva roky, konkurence mezitím spustí dvě totožné služby, jejichž uživatele bude velmi obtížné, ne-li nemožné, přetáhnout. [6, 14]

Někteří autoři[19] dokonce zavádějí v souvislosti s vývojem internetových aplikací pojem *Web-time*, aby tak zdůraznili vpravdě šílené pojetí času v tomto odvětví.

3.1.3.1 Agilní metodiky: kdo, kdy a proč

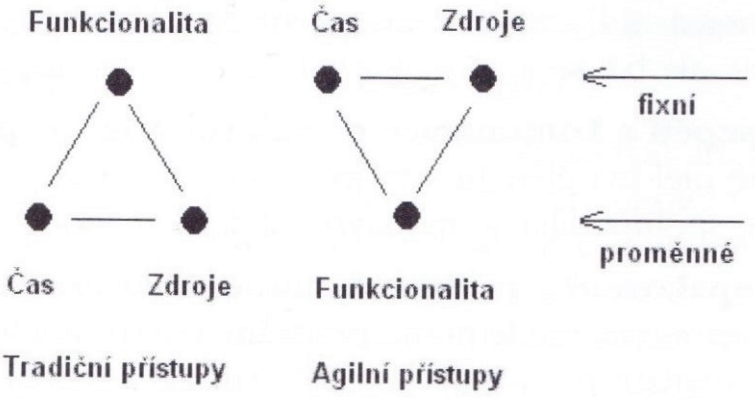
Vzhledem k důvodům popsaným výše se začátkem 3. tisíciletí nutně začínají prosazovat metodiky umožňující co nejrychlejší vývoj softwaru, jeho průběžnou údržbu a reakci na měnící se podmínky a zadání.

Tyto metodiky jsou nazývány **agilní**. Jejich původ pod tímto názvem nalézáme v roce 2001, v němž se sešli nejvýznamnější představitelé nových přístupů k tvorbě softwaru – včetně nejznámějších tváří – *Alistar Cockburn, Kent Beck, Ward Cunningham, Martin Fowler, Ken Schwaber, Jeff Sutherland* a další.[6]

Na setkání těchto osob se tedy stalo, co se muselo stát: po podrobné analýze jednotlivých metodik se překvapivě snadno shodli na základních zastřešujících tezích, z nichž posléze vzešla formulace Manifesti agilního vývoje[1].

Pojem **agilní metodika** tak označuje skupinu metodik vycházejících z poznání, že jedinou cestou, jak prověřit správnost navrženého systému, je co nejrychleji jej vyvinout, předložit zákazníkovi a na základě zpětné vazby upravovat.

Rozdíl mezi tradičními přístupy a agilními metodikami lze nejlépe pozorovat na následujícím obrázku, který srovnává vidění základních proměnných při vývoji softwaru.



Obrázek 1: Rozdíl mezi tradičními a agilními programovacími přístupy [6]

Tradiční metodiky vycházejí z nutnosti naplnit za každou cenu dokument Specifikace požadavků. Požadavky – tedy funkcionalita – jsou tedy fixní, zatímco v roli proměnných vystupují čas a potřebné zdroje. Agilní přístupy naproti tomu považují čas a zdroje za fixní

[1]Ještě předtím však založili Alianci pro agilní vývoj softwaru (The Agile Alliance) a následně až formulovali Manifest agilního vývoje (The Agile Manifesto), *https://www.agilealliance.org/* a *http://agilemanifesto.org/*

(stanovené na začátku projektu zadavatelem) a mění se (průběžně přizpůsobuje) funkcionalita (požadavky). [6, 14]

Základními principy agilních metodik jsou především:

- **Iterativní a inkrementální vývoj s velmi krátkými iteracemi**
- **Důraz na přímou, osobní komunikaci v týmu**
- **Nepřetržité sepětí a komunikace se zadavatelem, resp. uživatelem**
- **Rigorózní, opakované, průběžné automatizované testování**

Agilní metodiky vyžadují mnohem menší objem formálních a byrokratických artefaktů. Vycházejí přitom z neochvějného přesvědčení, že **základním, jednoznačným, exaktním, nezpochybnitelným a vlastně jediným spolehlivým nositelem informace je zdrojový kód**. [6]

Proto i agilní manifest vychází ze dvou základních tezí:

1. Přijmout a umožnit změnu je mnohem efektivnější než pokoušet se jí zabránit.
2. Je třeba být připraven reagovat na nepředvídatelné události, neboť ty bezpochyby nastanou: tento bod lze formulovat také tak, že jedinou jistotou je změna.

Z důvodů uvedených ve dvou tezích v předchozím odstavci dávají autoři manifestu přednost:

- **individualitám a komunikaci** před procesy a nástroji,
- **provozuschopnému softwaru** před obsáhlými, objemnými dokumentacemi,
- **spolupráci se zákazníkem** před uzavíráním mnoha smluv,
- **reakci na změnu** před striktním plněním plánu.

Na základě uvedených dvou tezí a čtyř preferenčních skupin lze formulovat dvanáct základních tezí agilních metodik:

- **Užitná hodnota pro zákazníka**
- **Změny jsou výhodou**
- **Časté dodávky**
- **Zákazníci spolupracují s týmem**

- **Motivace je klíčová**
- **Vzájemná konverzace**
- **Úspěch posuzujeme podle fungování**
- **Udržitelný vývoj**
- **Perfektní návrh, perfektní řešení**
- **Zásadní je jednoduchost**

V této souvislosti stojí také za zmínku tzv. princip Occamovy břitvy (Occam's razor): neměla by být předpokládána existence více věcí, než jsou absolutně nezbytné. Stejně hovoří agilní metodiky: děláme vždy jen naprosté minimum, které je nutné k dosažení cíle, nic navíc.

V současnosti se stále zvyšuje počet konkrétních metodik, které bezprostředně vycházejí z agilního manifestu. Následující přehled obsahuje jen nejvýznamnější zástupce. [6, 16]

- **Adaptivní vývoj softwaru** (Adaptive Software Development: Jim Highsmith)
- **Vlastnostmi řízený vývoj** (Feature-Driven Development: Jeff De Luca a Peter Coad)
- **Extrémní programování** (Extreme Programming: Kent Beck)
- **Lean Development**: Bob Charette
- **SCRUM Development Process**: Ken Schwaber a Mike Beedle
- **Crystal** metodiky (Crystal Methodologies: Alistar Cockburn)
- **Dynamic System Development Method** (DSDM Consortium)
- **Testy řízený vývoj** (Test-Driven Development: Kent Beck)
- **Kanban** (Toyota: Taiiči Óno)

3.1.3.2 Extrémní programování

Extrémní programování (Extreme Programming, XP) je pravděpodobně nejrozšířenější a nejznámější agilní metodikou: v očích mnoha lidí se jedná o jakéhosi představitele, základní ztělesnění agilních programovacích technik.

XP je označován jako účinný, efektivní, lehký, flexibilní, předvídatelný a zábavný způsob vyvíjení softwaru. Autor *Kent Beck* ve své knize [7] uvádí, že řadě lidí, kteří se s XP setkali,

se tato metodika jeví jako velmi rozumný, realistický způsob smýšlení, který bývá často označován jako „zdravý rozum“.

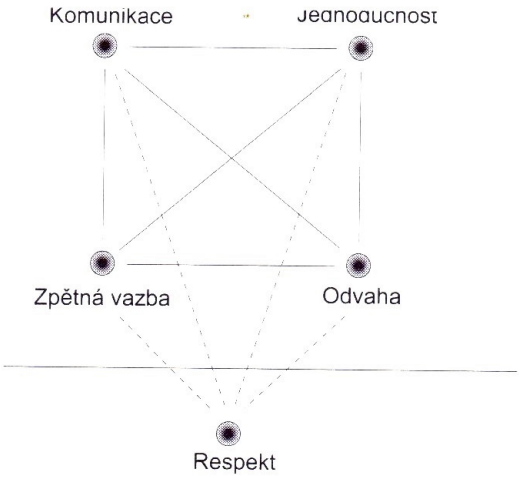
XP sice opravdu využívá běžně známé postupy a myšlenky, avšak jejich používání dotahuje do extrémů. [6, 14]

- **Jednoduchost**: V XP se vývojový tým vždy snaží vytvořit a dodat nejjednodušší možnou verzi, která ještě může fungovat.
- **Revize a kontroly zdrojového kódu**: Pokud se osvědčují, budeme kód revidovat a kontrolovat průběžně (neustále). Vede k *párovému programování*.
- **Návrh**: Protože se osvědčuje, budou všichni každodenně navrhovat a vylepšovat stávající návrh. Vede k *refaktorizaci*.
- **Architektura**: Je jedním z klíčových fragmentů projektu.
- **Testování**: Jestliže se osvědčuje, tak budou testovat všichni, i zákazníci, protože nikdo nedokáže přesněji určit, zda software dělá to, co se od něj vyžaduje. Vede k *unit testing* a testování funkcionality.
- **Krátké iterace**: Velice krátké iterace v sekundách, minutách a hodinách místo týdnů, měsíců a let.

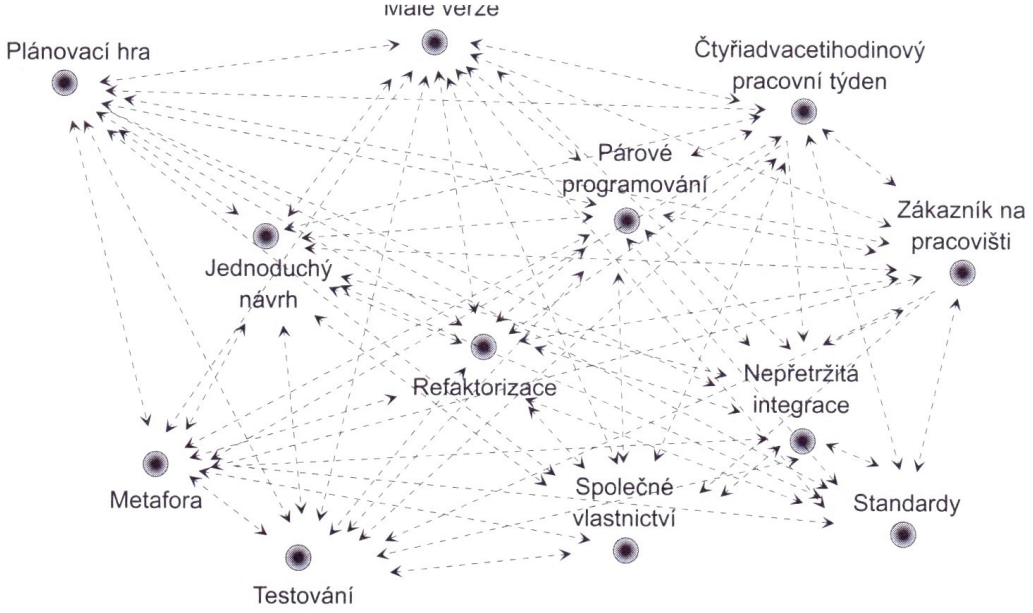
XP dále pracuje se čtyřmi proměnnými:

- kvalita
- čas
- náklady
- šíře zadání

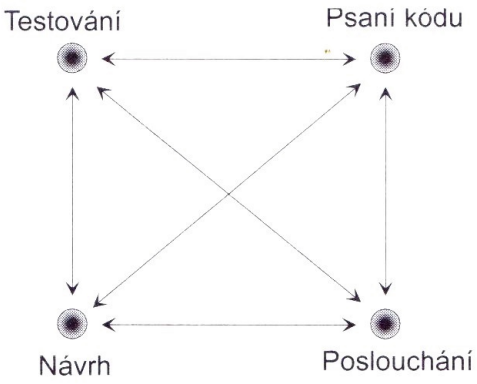
Důležitý axiom přitom zní: **zákazníci, zadavatelé, manažeři stanovují hodnoty libovolných tří proměnných; výslednou hodnotu zbývající, čtvrté proměnné pak vybere vývojový tým.**



Obrázek 2: Vztah pět základních hodnot v Extrémním programování [6]



Obrázek 3: Vzájemný vztah dvanácti postupů v Extrémním programování [6]



Obrázek 4: Vzájemný vztah činností v Extrémním programování [6]

Výhodou a silnou stránkou XP je práce v souladu s lidskými instinkty a nikoli proti nim: lidé se rádi učí, vyhrávají, stýkají s jinými lidmi, jsou součástí týmu, mají kontrolu, když se jim věří, odvádějí dobrou práci, když software funguje, když vidí blízký a konkrétní cíl. Další výhody plynou z iterativního a inkrementálního vývoje. Nevýhodou jsou pak obtíže při jeho praktické realizaci, především v době, kdy si vývojový tým na XP teprve zvyká – dělat věci jednoduše a naučit se svět vidět v nejjednodušších pojmech. [6]

3.1.3.3 SCRUM Development Process

SCRUM je agilní metodika vývoje softwaru, jejímž cílem je především zvýšení efektivity při vývoji softwaru. Podobně jako další agilní postupy i SCRUM využívá iterativního a inkrementálního přístupu. Inklinuje k objektově orientovanému vidění světa, ale organizuje a řídí proces zcela novým způsobem.

SCRUM vychází z objektově orientovaného přístupu, díky němuž odpovídá každý vývojář za množinu objektů s jasně definovaným chováním a rozhraním. Vývoj softwarového vybavení pomocí metodiky *SCRUM* probíhá v rámci tří až osmi posloupností pevně daných časových intervalů. Tyto intervaly se nazývají sprinty, *Sprints*. Každý z těchto sprintů obvykle trvá zhruba měsíc. Sprinty jsou tedy základním členění vývojového procesu podle metodiky *SCRUM*.

V rámci sprintů nedefinuje *SCRUM* žádné konkrétní procesy, předpokládá však denní schůzky (*Scrum meetings*), z nichž vzejde konkrétní určení činností. Na těchto schůzkách se shrnuje, které položky byly od minulé schůzky dokončeny a které nové úkoly byly nalezeny.

Každý sprint končí předvedením výsledku (*Demo*), který dá zákazníkům přehled o dění na projektu, vývojářům pocit dokončení úkolu a který umožní provést integraci a testování. [6, 14]

Projekty vyvíjené pomocí metodiky *SCRUM* mají několik základních charakteristik. Jejich společným jmenovatelem je flexibilita a spolupráce:

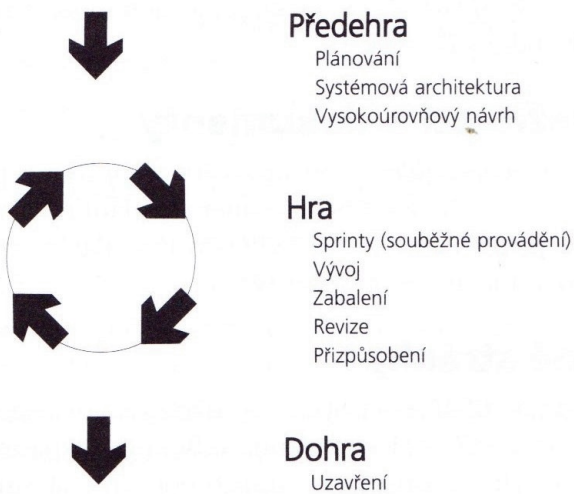
- **Flexibilní předměty dodání** – obsah dodávky je diktován prostředím
- **Flexibilní harmonogram** – dodání může proběhnout dříve nebo později
- **Malé týmy** – mezi třemi a šesti členy
- **Časté revize** – dosažený pokrok a případné změny jsou předmětem každodenního zkoumání
- **Spolupráce** – napříč vývojovým týmem i vzhledem k okolí

Důležitými pojmy metodiky jsou také *backlog, riziko, sprint* a *scrum meeting*:

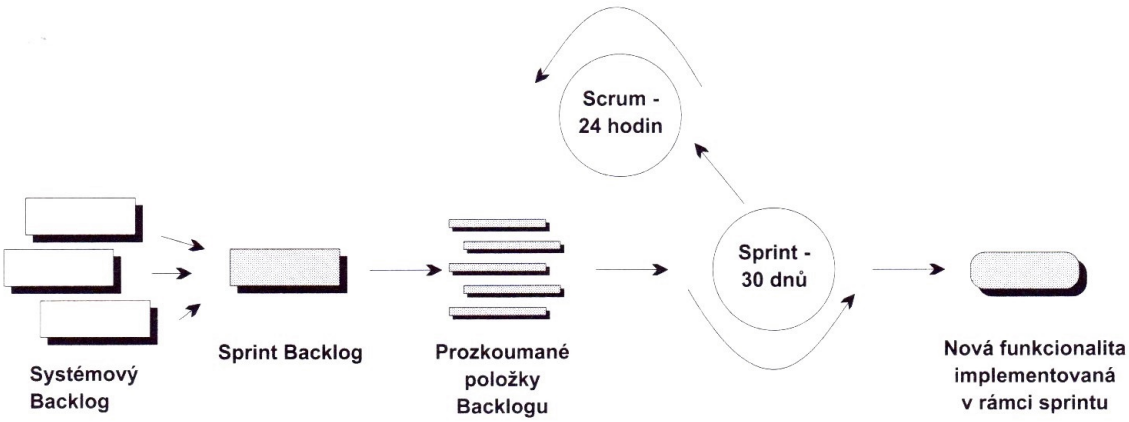
- **Backlog:** základní nosič informace o funkcích či vlastnostech, resp. činnostech, které je nutné implementovat, resp. provést.
- **Riziko:** metodika silně dbá na analýzu rizik; jsou revidována na konci každé iterace, ale i v průběhu sprintu či v rámci každodenních schůzek.
- **Sprint:** základní vývojová entita (iterace); obvykle probíhá 30 dní.
- **Scrum Meeting:** každodenní setkání celého vývojového týmu sloužící k přehledu vykonané práce, plánu příštího dne a k pojmenování všech důležitých aktuálních témat (rizik, změn, apod.).

Podíváme-li se pak na procesy ještě podrobněji, zjistíme, že SCRUM definuje čtyři vývojové kroky. První dva formálně náleží do tzv. **fáze předehry** (*Pregame*), třetí do **fáze hry** (*Game*) a čtvrtý do **fáze dohry** (*Postgame*):

- **Plánování** (*Planning*): definuje rozsah aktuální verze, harmonogram, nutné zdroje apod.
- **Architektura a design** (*Architecture/Design*): vytvoření nebo modifikace architektury v závislosti na nových požadavcích, poznatcích, rizicích
- **Vývoj** (*Development, Sprint*): iterativní cyklus vývojových prací



Obrázek 5: Metodika SCRUM Development Process [6]



Obrázek 6: Třetí fáze (Vývoj) metodiky SCRUM Development Process [6]

- **Uzavření** (*Closure*): příprava produktu k finálnímu uvolnění a šíření

Hlavní výhodou metodologie *SCRUM* je schopnost pružně reagovat na změny vznikající v průběhu práce na projektu. Metodika každodenně reflektuje případné změny a hledá rizika, která z nich vyplývají. Dále pak poskytuje vývojářům svobodu navrhnout optimální řešení, případně změnit přístup v průběhu projektu podle toho, kam se ubírají aktuální požadavky zákazníka.

Nevýhoda pak spočívá především ve skutečnosti, že se jedná spíše o souhrn vzorů než o specifikaci konkrétních kroků vedoucích k vývoji softwaru. Další nevýhody mohou být podobné jako v případě XP: obtížný přechod na nový způsob práce, nutnost najít vhodné

typy lidí a nutnost vyrovnat se s odlišným pojetím vývoje. [6, 14, 19]

3.1.3.4 Lean Development

Metodika *Lean Development* se od ostatních metodik odlišuje (kromě dalších parametrů) především způsobem svého vzniku: zatímco ostatní metodiky byly vytvořeny přímo s úmyslem formalizovat vývoj softwaru, *Lean Development* má své kořeny v jiných inženýrských (a především výrobních) disciplínách. Myšlenky převzaté z výrobních odvětví byly pouze převzaty a přizpůsobeny pro vývoj softwaru. Jedná se v zásadě o několik základních pravidel, jejichž aplikace do oblasti vývoje softwaru je podrobně vysvětlena a okomentována. [6, 16]

Podíváme-li se na metodiku *Lean Development*, uvidíme především několik klíčových pravidel, jejichž dodržování by mělo vést k optimálnímu, efektivnímu a kvalitnímu vývojovému procesu. [16]

Lean Development je zaměřena strategičtěji než ostatní agilní metodiky. Klade si také neskromné cíle:

- vyvíjet software za třetinu obvyklého času,
- vystačit s třetinou obvyklého rozpočtu,
- snížit četnost chyb na třetinu obvyklého množství.

Lean Development není exaktní metodika popisující, jakým způsobem vyvíjet a jak přesně řídit vývojový tým, nabízí však řadu principů a pravidel, jejichž dodržování povede k efektivnějšímu a k ekonomičtějšímu výsledku.

Jak již bylo uvedeno, metodika *Lean Development* je postavena na deseti obecných pravidlech, která byla definována už pro systém *Lean Manufacturing*.

1. Odstranit vše, co je zbytečné.
2. Minimalizovat zásoby (minimalizovat meziprodukty).
3. Maximalizovat tok (zkrátit čas potřebný pro vývoj).
4. Vývoj je „tažen poptávkou“ (většina rozhodnutí se dělá tak pozdě, jak je to jen možné).
5. Pracovníci mají pravomoc rozhodovat (rozhodování probíhá i na nejnižších úrovních).

6. Hlavním cílem je uspokojovat požadavky zákazníků (a to nejen teď, ale i v budoucnu).
7. Zavést zpětnou vazbu (nebát se změn v učiněných rozhodnutích).
8. Odstranit lokální optimalizaci (neustálé optimalizace stávajícího řešení postrádají smysl).
9. Vybudovat partnerství s dodavateli (využívat subdodávek a předpřipravených komponent).
10. Vybudovat kulturu pro možnost neustálého zlepšování.

Obrázek 7: Sedm principů v metodice Lean Development [6]

Lean Development by mohla být rozebrána a rozpracována podrobněji, nicméně celá spočívá vlastně v oněch deseti pravidlech a sedmi principech s tím, že každé pravidlo a princip by mohly být dalekosáhle rozebírány mnohem podrobněji.

Přesto však nelze uvažovat o metodice *Lean Development* jako o striktním, komplexním popisu vývoje softwaru. Metodika – stejně jako řada dalších agilních metodik – pouze naznačuje, jakým způsobem by měl vývoj probíhat, jak by měl být prováděn, čeho by se měl vyvarovat a kam by měl směřovat. *Lean Development* neříká nic o tom, že „ve druhém týdnu by měl pracovník v roli systémového analytika dokončit analýzu problému a vytvořit příslušný specifikační dokument, který předloží pracovníkovi v roli šéfa projektu ke schválení“. *Lean Development* pouze na deseti jednoduchých pravidlech popisuje současný svět a navrhuje některé metody, které by měly být při vývoji používány. [6, 16]

3.1.3.5 Test Driven Development

Jak už napovídá samotný název, metodika *Test-Driven Development* klade důraz především na testování vyvíjeného softwaru a staví testování na samotný vrcholek pomyslné pyramidy jednotlivých kroků při vývoji softwaru.[2]

V tradičních metodikách bylo testování zařazeno jako jedná z fází vývojového procesu, v agilních metodikách je testování často prováděno průběžně jako nedílná součást všech ostatních aktivit při vývoji. Až dosud jsme se však nesetkali s metodikou, která by prohlásila testování nikoliv za *jednu z několika* vývojových fází, ale za *základ* celého procesu.

Výše uvedenou myšlenku prosazuje *Test-Driven Development*. Tato metodika totiž říká: „Pro každou drobnou součást funkcionality ve zdrojovém kódu je nutné nejprve napsat test, který dokáže příslušný programový kód dokonale otestovat a ověřit.“ Jinak řečeno *Test-Driven Development (TDD)* vyžaduje napsání testu *ještě před* napsáním kódu, který se má testovat.

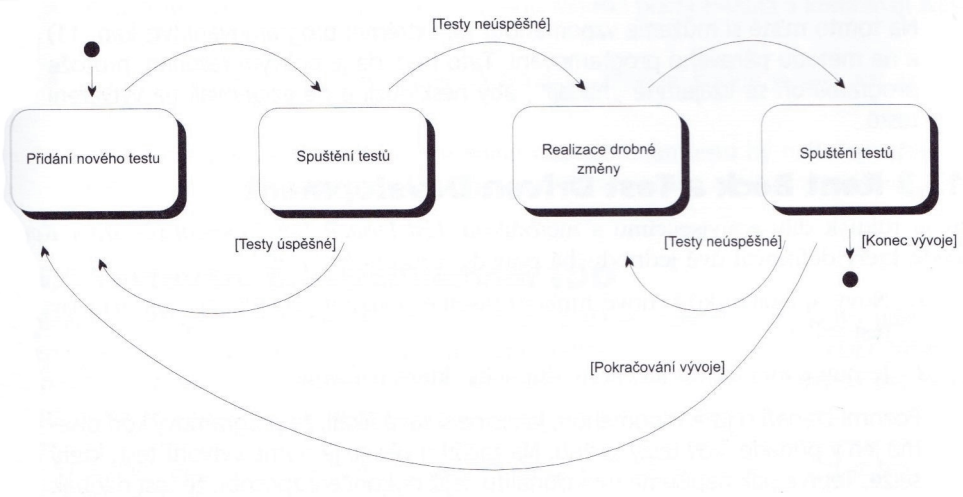
Teprve když máme hotový testovací kód, naprogramujeme samotnou funkčnost, přičemž dodržujeme pravidlo, že *implementujeme přesně takové množství programového kódu, kolik dokáže projít testem*. Ne méně, ale také ne více.

Po dokončení zdrojového kódu, který projde testem, nastává fáze úprav: prostřednictvím refaktorizace (úprav zdrojového kódu bez změny funkčnosti) zjednodušujeme a zefektivňujeme jak zdrojový kód, tak kód testu. [6, 14]

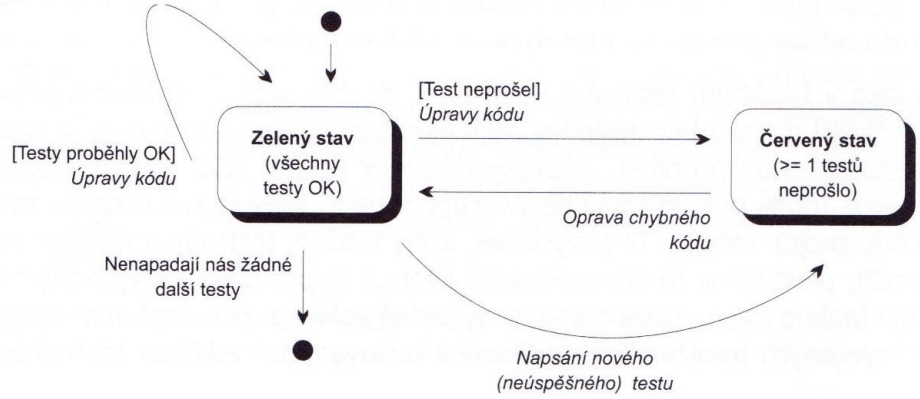
Jedná se svým způsobem o revoluční přístup k vývoji. První činností v rámci *TDD* je napsání testu; to však není všechno. *TDD* říká, že musíme napsat takový test, který selže. Teprve když máme selhávající test, můžeme implementovat testovanou funkci: její implementace je u konce ve chvíli, kdy funkce projde tímto testem.

Mnoho programátorů, pravděpodobně jich je většina, nemiluje dvakrát čtení dokumentací a specifikací. Pokud potřebují zjistit, co dělá nějaká třída nebo metoda, mnohem raději nejprve sáhnou po ukázkovém příkladu, který ukáže daný kousek kódu za běhu.

[2] Testování je jedním z nejviditelnějších a neznámějších složek oblasti *Quality Assurance (QA)*, tedy zajišťování kvality. Již méně se ví, že testování není zdaleka nejúčinnější metodou: daleko lepších výsledků (pokud jde o procento nalezených chyb) dosahují jiné složky QA, jako například strukturované procházení, faganovské inspekce, revize návrhů apod.



Obrázek 8: Grafické znázornění metodiky Test-Driven Development [6]



Obrázek 9: Obecný přístup testování prostřednictvím TDD [6]

Testovací moduly a jednotky ve své podstatě představují přesně totéž: poskytují návod, jak používat testovací kód, a zároveň ukazují jeho specifikaci. Lze tedy říci, že kvalitně vytvořené testovací případy se mohou stát součástí technické dokumentace produktu, protože do značné míry zastupují funkční specifikace a další důležité dokumenty.[3]

Zastánci metodiky *TDD* říkají, že vyvíjet software tímto způsobem je možná nejefektivnější metoda. Ptají se totiž: proč mám plýtvat energií a produkovat chyby předtím, než je opravím? Lepší je napravit je, dokud jsou „čerstvé“, dokud se zabýváme místem jejich

[3] Analogikcy lze dovodit, že akceptační testy mohou být zase použity jako součást specifikace požadavků. Akceptační testy totiž přesně definují, co zákazníci od systému očekávají, stejně jako v případě, kdy nám diktují své nejdůležitější požadavky.

výskytu a dokud si s nimi nejlépe dokážeme poradit.

Marcus Baker, který se zabývá vývojem pomocí *TDD* už několik let, říká, že pokud je v jakémkoliv rozhovoru tázán na svou představu o životním cyklu, odpovídá jednoznačně: nejprve testování, pak implementace a pak návrh. Zároveň dodává, že tímto způsobem lze z vývoje softwaru zcela eliminovat známou, nepříjemnou fázi hledání chyb (*debugging*): při použití *TDD* zkrátka debugging neexistuje, protože veškeré chyby (*bugs*) jsou odchyceny mnohem dříve, hned při průchodu testovacím případem. [6, 14, 17, 11]

3.2 Dnešní pohled na tvorbu frontendu

Cílem dobrého webdesignera je vytvořit web, který bude fungovat – tedy plnit svůj účel pro byznys klienta a naplňovat potřeby návštěvníků webu. V praxi to znamená, že webová prezentace má pro klienta jasný přínos, který je nejčastěji reprezentován finančním ziskem. Díky webu klient peníze vydělá nebo ušetří. Peníze jsou zásadní metrika, která říká, že jste vyřešili klientův problém. Návratnost investic může mít různé formy v závislosti na tom, kde web stojí v rámci strategie klienta. Potřebě přínosu by měla být podřízena i snaha webdesignera. Web musí vydělat klientovi peníze, aby se vrátily náklady spojené s tvorbou webu a web vytvářel zisk. Webdesigner si musí být vědom toho, že jeho výtvor nebude nakonec měřen estetickou kvalitou, ale faktickým výkonem[4] dle předem nastavených metrik. Úspěchu nelze dosáhnout, pokud se někdo soustředí jen na grafické ztvárnění webu. Výhodou tohoto přístupu je dlouhodobá spolupráce s klientem, která přináší další práci, takže vydělají oba. Klient, kterému web funguje, nemá důvod jít jinam. [5, 3]

3.2.1 Typy webových projektů

Weby můžeme z pohledu požadovaného výkonu rozdělit do tří základních skupin:

- Webová prezentace má za cíl ovlivnit či změnit chování určité skupiny lidí – prezentuje určitý produkt nebo službu a je často kanálem pro prodej.
- E-shop prodává produkty či služby online – cílem e-shopu není prezentace produktů, ale především jejich přímý prodej.

[4] Estetická kvalita výkon webu podporuje, ale nepředstavuje cíl, nýbrž prostředek.

- Webová aplikace řeší určitý problém svých uživatelů prostřednictvím sebe sama. Není kanálem pro prodej produktu, ale přímo produktem. Cílem designera webových aplikací je vytvořit nový návyk – aplikace tedy zapadne do života člověka, který ji používá.

Obrázek 10: obsah – interakce [5]

Pro tvorbu webových prezentací jsou potřeba jiné znalosti a dovednosti než pro tvorbu webových aplikací. Webové prezentace musí umět prodat to, co propagují. U webových aplikací jsou klíčové dobře navržené interakce, snadnost používání a schopnost aplikace podpořit konkrétní procesy klienta nebo jeho zákazníků. [5]

3.2.2 Kdy weby fungují?

Web, který funguje, vytváří přínosy, a tedy přináší klientovi zisk. Ten ale nemusí přijít hned. Přínosy má web především díky třem faktorům:

- Lidé na něm provedou konverzní akci. Může se jednat o odeslání objednávky, sdílení stránky na sociálních sítích, registraci do newsletteru ...konverzní akce je často i jednoduše měřitelná.
- Lidé na něm naleznou informaci – člověk na webu najde to, co hledá, informaci zkonzumuje a odchází. Možná na základě této informace začne v budoucnu přemýšlet o tom, že potřebuje produkt klienta, ale jedná se o dlouhodobější proces. Samotná konzumace se měří výrazně hůře než konverzní akce, ale mnoho webů je postaveno právě na schopnosti předat správným způsobem informace. Díky webu se pak snižuje zátěž call centra, postupně se buduje databáze potenciálních zákazníků, zvyšuje se návštěvnost a nakonec i přímé konverze.
- Lidé na základě webu získávají pocit – web vyvolá emoce, které si návštěvník spojí s brandem provozovatele webu. Díky pocitu si návštěvník web snadněji zapamatuje,

bude o něm spíše mluvit s ostatními a v budoucnu se může vrátit kvůli informacím, potažmo konverzní akci. [5, 3]

3.2.3 Kdy weby selhávají?

Při mé dosavadní praxi jsem se setkal s těmito základními faktory neúspěchu webového projektu, který způsobil webdesigner:

Schopnosti webdesignera

Webdesigner není schopen doručit výstup v odpovídající kvalitě a projekt selže na jeho neschopnosti. Problém může nastat na strategické úrovni, kdy je naplánován nesmyslný web, který neosloví cílovou skupinu. Nebo může vzniknout problém na úrovni exekuce, kdy je web pomalý, nepoužitelný či nedůvěryhodný. V obou případech je projekt zahozen nebo živoří.

Komunikace

Špatná či nedostatečná komunikace má za následek vzájemné nepochopení webdesignera a klienta. Patří sem i neznalost korporátní politiky na straně webdesignera. Selhání skvělého projektu může nastat například ve chvíli, kdy se odnikud vynoří osoba na straně klienta, která v projektu dosud nebyla zainteresována, a svou politickou silou ho zastaví nebo výrazně zkomplikuje.

Projektové řízení

Webdesigner není schopen dodat výstup v termínu za dohodnutou cenu. Projektové řízení zvládá málokdo a je mu věnována část této práce.

Marketing

Část klientů se ke spuštění webu upne jako k zásadnímu milníku, který vyřeší všechny jejich problémy. Tato představa je falešná. Web je jedním z prvků marketingové strategie a bez marketingu nebude nikdy fungovat. [5, 14]

3.2.4 Intuitivní proces návrhu webu

V době psaní této práce používá mnoho webdesignerů[5] následující přístup k návrhu webu:

- Zjistí od klienta, co chce na webu mít

[5] Část ne, ale většina, s kterými jsem se kdy setkal, ano.

- Přidá své vlastní podněty
- Navrhne wireframe[6] titulní strany a načrtne mapu webu[7]
- Klient vše schválí a webdesigner pokračuje grafickým návrhem titulní strany
- ...

Tento postup je rychlý, levný a v rukou většiny je zároveň tragicky nefunkční. Cílem této práce je změnit přístup realizátora k návrhu webu. Proč není doporučeno používat intuitivní proces?

1. Klient netuší, co by mělo být na webu – je to laik, nikoli expert na návrh webu (ačkoliv si vždy něco vymyslí).

2. Jestliže není jasná představa o klientově byznysu a jeho zákaznících, jsou podněty víceméně nepoužitelné.

3. Wireframe titulní strany webu je k ničemu. Web funguje jako celek a titulní strana slouží v množství případů k tomu, aby z ní lidé odešli[8]. Je to tedy jedna z méně důležitých částí webu, ačkoliv jako jediná bude mít rozmyšlený obsah a jeho rozvržení.

4. Klient wireframe a mapu webu schválí jen proto, že netuší, že tvorba nemusí dopadnout dobře – přesvědčí se o tom až ve chvíli, kdy je web těsně před spuštěním a z daleka nesplňuje ani to, co by po něm chtěl, natožpak aby měl nějaký faktický obchodní výkon.

Intuitivní proces má i různé varianty – včetně té, kdy se vytváří wireframy celého webu, které se ale nikdy nedostanou k otestování ani nejsou postaveny na potřebách skutečných návštěvníků. Webdesigner je plně zodpovědný za proces, kterým se dostane k výsledkům, a pokud využívá intuitivní proces návrhu webu, tak k tomu musí mít jednoznačný důvod a musí klientovi vysvětlit důsledky (tedy, že to možná nebude fungovat a lze tomu předejít). Návrh webu na první pohled vypadá jako obor, ve kterém je potřeba hlavně selský rozum, kterým se nahradí skutečné znalosti a dovednosti. Webdesigner nedojde pouze se selským rozumem daleko. Intuitivní proces je přímým produktem selského rozumu a dokazuje, že skutečnou intuici je potřeba systematicky budovat. [5, 2]

[6] Drátěný model

[7] Hierarchický seznam stránek, které na webu budou

[8] Tam, kam chceme... ale odešli.

3.2.5 Struktura a obsah webu

Širší kontext webu začněte převádět v podrobnější popis řešení (stále ne nutně graficky). Osobně mi pomáhá dát všechny informace na jedno místo (v mém případě je to myšlenková mapa – ve vašem to můžou být třeba lístečky a poznámky na stěně) a přemýšlet nad nimi v jednom velkém celku. Z nich pak postupně vzniká struktura obsahu (a u ní poznámky ke konkrétním funkcím webu), kterou dále rozpracovávám. Co vám může pomoci při tvorbě obsahu?

3.2.5.1 Obsahová strategie

Obsahová strategie se zabývá plánováním, tvorbou a vyhodnocováním účinnosti obsahu napříč všemi marketingovými kanály, mmj. i webem tak, aby naplňoval potřeby klienta a jeho zákazníků. Typicky obsahuje:

1. zmapování aktuálních kanálů, na kterých vytváříme obsah,
2. zmapování aktuálního webu (pokud redesignujeme),
3. stanovení principů pro tvorbu obsahu a procesů pro práci s obsahem,
4. naplánování vytvoření a převodu stávajícího obsahu,
5. naplánování tvorby dalšího obsahu na jednotlivých kanálech,
6. nastavení a vyhodnocení KPI.

Díky obsahové strategii máte obsah pod kontrolou a zachováváte jeho konzistenci, aktuálnost, relevantnost, apod. Obsahový stratég je něco jako analytik a šéf lidí, kteří vytvářejí obsah, v jedné osobě. Udává směr a vyhodnocuje výsledky. Osobně používám pro návrh obsahové strategie webů framework Avinashe Kaushika See-Think-Do[9] formou myšlenkové mapy. Obsahovou strategii podrobně rozebírají knihy na konci kapitoly. Pokud chcete navrhovat webové prezentace, musíte znát základy obsahové strategie (u webových aplikací je naopak moc nevyužijete). [5]

[9] http://naostri.se/kaushikSTD

3.2.5.2 Designové frameworky

Jedním z prvních kroků k návrhu obsahu webu pro vás mohou být designové frameworky. Ty vám dají návod, jak přemýšlet nad návrhem obsahu jednotlivých prezentačních stránek. Chcete, aby vám lidé na určité stránce vyplnili e-mailovou adresu? Tu vám ovšem bez jasného a srozumitelného přínosu skoro nikdo nedá. Proč by se měl váš návštěvník nechat spamovat? Pro získání e-mailu můžete použít framework Registrace:

1. Uvedete jasný a viditelný nadpis, který definuje, co návštěvník získá, když vám poskytne e-mail (Získejte ZDARMA 50 stránkové kompendium hojení ran).

2. Rozvedete zásadní konkrétní výhody toho, proč vám má dát e-mailovou adresu (udělá si přehled o tom, jak hojit rány, o prevenci, správné stravě pro podporu hojení a 1 krát měsíčně mu pošlete odkazy na zajímavé články okolo hojení ran).

3. Přidáte poděkování stávajícího odběratele pro zvýšení důvěryhodnosti (s fotografií, celým jménem a uvedením věku dle demografie cílové skupiny webu).

4. Uvedete jasnou a viditelnou výzvu k akci (formou výzvy a nikoliv v trpném rodě – získejte kompedium již dnes).

5. Vše podpoříte grafikou (např. obrázek knihy – kompendia).

Konverzní poměr s použitím výše uvedené struktury informace bude mnohonásobně vyšší, než kdybyste dali na web pouze formulář. [5, 3, 14]

3.2.5.3 Návrh informační architektury

Návrh informační architektury a navigace webu je samotný a rozsáhlý obor – nejlepší je začít knihou od pánů Morvilleho a Rosenfelda uvedenou níže. Pár tipů ode mne:

- Používají klienti a návštěvníci webu stejný jazyk?
- Jaký obsah potřebujete na webu mít?
- Podle jakých kritérií budete seskupovat obsah webu?
- Podle jakých kritérií budete řadit obsah webu?

- Budou lidé často přecházet mezi různými kategoriemi?
- Jak lidé hledají obsah dnes? Budete mít k dispozici klasifikační analýzu klíčových slov? [5]

3.2.5.4 Pozice webu v cestě uživatele

V tuto chvíli byste už měli vědět, jaká bude pozice webu v byznysu klienta. V praxi to znamená, že na něj odněkud budou chodit návštěvníci a na základě své činnosti na webu budou pokračovat na další místa, kde se mohou potkat s klientovým brandem. Na webu pravděpodobně budou probíhat nějaké konverzní akce, které budou mít pro klienta přínos jen v případě, že jim budou předcházet jiné aktivity a akce, které na webu neovlivníte. Proberte je s klientem a ukotvěte web vzhledem k jeho pozici v klientově marketingovém mixu. Pomůže vám to lépe naplánovat průchody uživatelů webem a přinesete klientovi další přidanou hodnotu tím, že ho upozorníte na možná úskalí jeho služby (např. že potřebujete mít rozpočet na skvělý hromadný e-mailingový systém, protože jinak je mu sbírání e-mailů k ničemu). [5]

3.2.5.5 Průchody návštěvníka webem

Smyslem této metody (anglicky task flows) je zmapovat, jak budou lidé procházet webem. V případě, že je k dispozici uživatelský výzkum, dojde k rozpracování motivace návštěvníků do konkrétních sekvencí stránek.

1. Lidé budou mít určité motivace, se kterými na web přijdou a které jste už zmapovali v rámci výzkumu.

2. Lidé přijdou z externího marketingového kanálu na konkrétní stránku webu, pravděpodobně to nebude úvodní strana webu.

3. Lidé projdou určitou sekvencí stránek – důležitá zde není hierarchie, ale návaznost stránek mezi sebou vzhledem k cíli návštěvníka.

4. Návštěvníci v ideálním případě provedou nějakou konverzní akci.

Díky task flows budete schopni přemýšlet o širším mentálním nastavení návštěvníka při návrhu obsahu jednotlivých stránek webu. Některé návaznosti stránek pochopíte intuitivně (průchod objednávkou v e-shopu), jiné navrhnete na základě výstupů z uživatelského výzkumu a obecných modelů chování lidí. [5]

3.2.5.6 Mapa obsahu webu

Mapa obsahu webu (Website Content Map) je moje vlastní metoda, jak si ulehčit přemýšlení nad obsahem webu. Vytvořte si mindmapu a začněte do ní přidávat stránky, které by z vašeho současného pohledu měl web mít. Stránka je velmi nepřesný pojem (nemusí to být nutně samostatné URL) – řekněme, že jde o sekce, které se budou na webu vyskytovat a některé z nich budou zároveň i samostatné stránky. Ke každé stránce si udělejte poznámku.

- Cíl stránky,
- co chceme návštěvníkovi říct – jakou pro něj máme na stránce zprávu,
- co by měl návštěvník na stránce udělat.

Příklad: O nás

- Cíl: Podpořit důvěryhodnost služby v očích zákazníka.
- Zpráva: Máme za sebou skoro 100 let vývoje, víme, co děláme.
- Akce: Přejít na ohlasy spokojených zákazníků.

Jestliže dáte každé stránce smysl, jednoduše odfiltrujete všechny, které smysl nemají nebo na něj nemůžete přijít. Následně doplňte k jednotlivým stránkám obsah, kterým zařídíte, aby byl splněn cíl stránky, návštěvník získal zprávu, kterou mu chcete předat, a provedl požadovanou akci. Obsah rozepište až na úroveň konkrétních slov, které se budou na stránce vyskytovat. Nestačí vědět, že na hlavní straně bude nadpis, potřebujete vědět, jaký je konkrétní nadpis. Mapa obsahu webu je místo, kde se stýká návrh webu a copywriting. Nejsem copywriter, ale dávám copywriterům zadání formou svých velmi konkrétních textů (a grafikům prostřednictvím proškrtnutých boxů... které ale opět velmi konkrétně popíšu). Copy writer pak může pracovat s velmi konkrétní představou, a nikoliv se zadáním á la „tady budou tři hlavní benefity“. To byste nemuseli poznávat klienta či návštěvníky webu! Mapa vám

pomůže definovat strukturu webu a konkrétní obsah jednotlivých stránek, který následně můžete vizualizovat pomocí skic a prototypu. Ale základ máte ujasněn a máte co kreslit. [5, 3]

3.2.6 Prototypování

Tvorba prototypu webu má tři fáze: skicování, wireframy a prototyp. V rámci prototypu navrhnete rozložení obsahu a jeho priority na jednotlivých stránkách webu. Ty pak vzájemně provážete. Je tedy možné projít klíčové interakce návštěvníka s webem a výstup otestovat. [5, 10]

3.2.6.1 Skicování

Skicování je levná a rychlá metoda, která vám pomůže generovat nápady. Vezměte si papír a tužku a začnete si kreslit. Obsah a smysl už máte ujasněn, teď je potřeba je převést do velmi hrubé vizuální podoby. Skicování vám umožňuje vyhnout se prvoplánovým nebo zcela chybným řešením. Udělejte si ke každé stránce webu několik skic. Skicujte v malém – třeba 6-9 skic na formát A4 a zkoušejte možná rozložení obsahu webu. Další velká výhoda skic spočívá v tom, že umožňují vyvářet nápady v týmu – skicovat může každý a všichni chápou, že skica je skica a nikoliv grafický návrh. Jestliže pracujete s kolegou grafickým designerem, skicujte spolu. Ukazujte skici ostatním členům týmu a probírejte s nimi rozpracované rozvržení stránek – pochopí ho spíše než mapu obsahu webu a dostanete zpětnou vazbu předtím, než se vrhnete na náročnější části procesu. [5, 1]

3.2.6.2 Wireframy

Cílem wireframů je rozvrhnout obsah jednotlivých stránek webu, které můžete odprezentovat klientovi, a on je pak může předat dodavateli svého webu. Skici můžete také ukázat klientům, ale bez slovního komentáře jsou prakticky nepřenositelné. Z drátěného modelu by mělo jít jasně vidět:

- Jaký obsah bude na dané stránce.
- Jakou bude mít vizuální prioritu.

- Jak bude rozvržen a jaké budou vztahy mezi jednotlivými částmi obsahu.

Wireframe je podklad pro grafickou interpretaci webu – čím zkušenější grafický designer je, tím více si může dovolit wireframe přeskládat a významněji ho interpretovat. Priority a obsah by ale měly být víceméně totožné. Uvažujte o wireframu jako o kostře, na kterou v budoucnu natáhnete kůži (grafický design) - což ale neznamená, že webový grafik vybarvuje vaše wireframy. Při návrhu wireframů už máte dostatečnou představu, jaký obsah bude na webu. Nemá tedy smysl používat výplňové texty typu lorem ipsum – raději použijte obsah stávajícího webu (či webu konkurence). Osobně píšu texty do jednotlivých wireframů, které pak slouží jako podklad pro copywritera (to nemá smysl u delších textů, tam při absenci jiných zdrojů využijete výplňové texty). [5, 1]

3.2.6.3 Prototyp

Slovo prototyp může mít různý význam. V kontextu této práce se jedná o wireframy provázané odkazy. Primárním cílem prototypu je vytvořit prostředek pro komunikaci s klientem nad budoucím webem. Prototyp si na rozdíl od wireframů může proklikat, vyskakují v něm skrytá menu, je možné odeslat formulář – je z něj tedy výrazně jasnější, jak se web bude skutečně chovat. Pro agenturní/freelance návrh webu je prototyp jedním z klíčových momentů, kdy se láme chleba, a dobře navržený a odprezentovaný prototyp může rozhodnout o realizaci celého projektu a naopak. Klienti nejsou v roce 2017 na proklikávací prototypy webů zvyklí, a tak můžete management klienta velmi pozitivně překvapit. A to až tak, že se někdy ani nezmůžou na slovo. Výhody prototypu jednoznačně převažují těch pár minut, kdy budete propojovat jednotlivé wireframy mezi sebou. Prototypování stojí na stejných principech, jako tvorba grafických návrhů, a proto byste měli mít alespoň základní znalosti grafického designu. Bez nich se může snadno stát, že grafičtí designeři budou muset zcela překopat vaši ideu a budou s vašimi výstupy pracovat jen s krajní nechutí. Úroveň interaktivity prototypu je plně závislá na jeho účelu – pro prezentaci klientovi stačí některé části popsat slovně (u ikonky se objeví po najetí myši vysvětlující text), pokud budete chtít prototyp podrobit uživatelskému testování, vyplatí se dotáhnout jeho obsah a funkčnost alespoň do takové podoby, aby vám testování k něčemu bylo (lidé budou při testování mnohem citlivější na výplňové texty než váš klient). Prototyp je potřeba po vytvoření vyvážit se

všemi informacemi, které o projektu do této chvíle máte, a okomentovat ty části, které jste se rozhodli neprototypovat. Prototyp je jen částečný pohled na řešení a vždy vyžaduje trochu představivosti.

- Porovnejte prototyp se záměrem klienta – má pro klienta smysl?
- Projděte si dokumentaci uživatelského výzkumu a svoje poznámky – odpovídá prototyp očekávání návštěvníků webu?
- Porovnejte jednotlivé stránky prototypu mezi sebou – jsou tam opakující se vzory, nebo je to každý pes jiná ves?

Následně si dejte pauzu a prototyp si projděte ještě jednou za pár dnů. Díky odstupu najdete pro některá sporná místa lepší řešení. Prototyp nikdy neposílejte e-mailem klientovi, který není na tento postup z vaší předchozí spolupráce zvyklý. Doporučuji prototyp prezentovat vždy osobně. [5, 1, 10]

3.2.6.4 Mobile First

Při tvorbě wireframů a prototypů se mi osvědčila metodika Mobile First a současné vytváření mobilní, tabletové a desktopové varianty každé stránky. Ušetříte si tím spoustu práce oproti případu, kdy navrhujete pouze desktopovou variantu a mobilní verzi necháte na grafikovi či kodérovi a zároveň vytvoříte jednodušší a použitelnější weby i pro desktop. [5]

3.2.7 Grafický návrh webu

Při návrhu vizuální podoby webu (tedy toho, čemu všichni obvykle říkají design) je nutné udržet své ego na uzdě a přemýšlet. Je velice jednoduché sklouznout k tomu, že člověk navrhuje web, který se mu líbí, nikoliv web, který bude fungovat. Líbivost je poplatná trendům, nehledí na rámec klienta, projektu či lidí, kteří budou na web chodit. Mnoho lidí má pocit, že grafický design je subjektivní a každý na něj má vlastní názor. Grafický design ale není o názoru, ale o tom, zda podporuje přínosy projektu či nikoliv. Je velmi vhodné, aby se klient s výsledkem ztotožnil (koneckonců ho platí), ale nečekejte od něj jinou zpětnou vazbu než líbí/nelíbí. Váš klient není odborník na webdesign, může jít o jeho první či druhý web v životě a nemá ponětí o tom, zda je vámi dodaný výstup dobrý nebo špatný. Pokud

budete hledat zpětnou vazbu na svou práci pouze u klientů, během několika let spláčete nad výdělkem – skutečnou zpětnou vazbu dostanete jen důsledným měřením faktického výkonu vašich projektů a/nebo prostřednictvím zkušenějšího webdesignera (částečně vám pomohou i marketéři nebo management, ale třeba z hlediska grafického designu jen webdesigner). Doporučuji vám tedy aktivně vyhledávat zpětnou vazbu – a pokud není v okolí nikdo tak dobrý jako vy, hledejte ji v zahraničí. Kooperaci se zkušenějšími doporučuji především proto, že si můžete velmi rychle uvědomit skutečné chyby webu, jejich příčiny a důsledky, a začnete se posouvat řádově rychleji než doposud. V rámci grafického designu existují prošlapané cesty, kterými je vhodné se ubírat před tím, než se vrhnete do křovisek a začnete se probíjet grafickou džunglí sami. Tyto cestičky jsou zdokumentované, popsané a můžete se od nich odrazit před vlastním bádáním. Dávám zpětnou vazbu grafickým designérům už několik let a setkávám se se stále stejnou neznalostí základních grafických principů. Pro grafický design potřebujete talent a trpělivost. Bez talentu se nikdy nedostanete na špici, na druhou stranu i bez něj můžete vytvářet vysoce funkční weby (z mého pohledu je talent 5 %, zbytek je píle). Klíčové je, že ve webdesignu nikdy nelze oddělit grafický design, interakční design a obsah webu – ten, kdo se o to pokusí, se stává pouhým dekoratérem. Proto jsou tvůrci wireframů, kteří netuší o grafickém designu, řádově znevýhodněni oproti lidem, kteří si prakticky prošli tvorbou grafických návrhů. Stejně tak jsou tradiční grafičtí designeři znevýhodněni před lidmi, kteří ovládají interakční design a dokážou navrhnout obsah webu. Webdesign je obtížnější disciplína, než tradiční grafický design[10]. [5, 10, 1]

3.2.7.1 Grafický rámec projektu

Při každém návrhu webu vyjděte z kontextu, který web obklopuje. Typicky nastává situace, kdy:

- Klient má logotyp a grafický manuál.
- Existují letáky, katalogy, billboardy, vizitky, ilustrace, fotografie, barvy, rastry, motivy apod., které budou s webem vizuálně provázány.

Ideální je v takovém případě kooperovat úzce s grafickým designerem klienta a vytvořit web jakou součást celku. Klient už může mít nějaké weby. Pokud nemá být váš

[10] Plakáty, loga a tak vůbec.

web základem nového vizuálního stylu, je velmi vhodné vycházet při tvorbě nového webu i ze stávajících webů klienta. Neříkám kopírovat, říkám vycházet. Může se stát, že váš grafický návrh bude odmítnut, protože nesedne do klientova grafického rámce. Druhý extrém je, že klient neví nic, a začínáte na zelené louce. V takovém případě doporučuji nechat tradičního grafického designera položit základ klientova vizuálního stylu, a pak na jeho práci navázat a vzájemně kooperovat. [5]

3.2.7.2 Smysluplnost

Na začátku každého webového projektu by měla být položena jednoduchá otázka. Proč to vlastně celé děláme? Jaký smysl má naše snažení mít? Bez smyslu nebude mít projekt přínos i v případě, že bude jinak navržený skvěle. Odpovězte si na následující otázky:

1. Řeší váš web něčí problém?
2. Pálí daný problém dostatek lidí?
3. Jsou za řešení ochotni zaplatit, aby se to vyplatilo?

Můžete vytvořit krásný web – pokud nebude uspokojovat potřeby určité skupiny lidí, je k ničemu. Smysluplnost je místo, kde se web stýká se skutečným světem. Obecně je web smysluplný ve chvíli, kdy je jeho potřeba přímým důsledkem klientovy strategie (opět narážíme na to, že ve chvíli, kdy klient nemá jasně definovanou strategii, je možné, že web děláte zbytečně). Smysluplný web má naději zapadnout do života klientových návštěvníků a pomoci jim splnit jejich životní cíle nebo alespoň aktuální potřeby. Smysluplnost je na jedné straně problém především vašeho klienta, na druhou stranu – vytvořte pár nesmyslných projektů za sebou a vcelku jednoduše vyhoříte. Tápete v tom, zda projekt bude chtít někdo používat? Zapomeňte na manažerský přístup typu vodopád a přejděte na některou z agilních metodik. Místo tvorby webu od začátku do konce potřebujete rychle a levně ověřit, že jsou vaše hypotézy správné. [5]

3.2.7.3 Dostupnost

Dostupný web je rychlý a nevidíte na něm na první pohled chyby – návštěvník se na něm tedy může pohybovat. Na dostupném webu nenajdete stránky „Error 404 – page not found“, a pokud už návštěvník narazí na chybu, je vysvětlena srozumitelným jazykem a rychle opravena.

Jen málokterý klient se podrobně zabývá tím, jak je navržena jedna z dražších částí jeho nového webu (po obsahu tvoří vývoj nebo customizace CMS[11] pravděpodobně nevyšší položku rozpočtu). Je to dáno tím, že programování nerozumí, takže musí důvěřovat expertům, které si najal. Bez zkušeného oponenta nemá šanci poznat, zda se jedná o bastl, nebo kvalitní, modifikovatelný, standardizovaný a otestovaný výsledek (a to často ani na straně dodavatele – account manažer neověří kvalitu kódů ač o ní bude klienta ujišťovat).

Klient špatně (a někdy i levně) naprogramovaný web nepocítí hned, ale z dlouhodobého hlediska se může zvláště u větších systémů, které se dynamicky rozvíjejí, jednat o zásadní milník pro úspěch projektu. Od systémové části webu se očekává, že bude rychlá, bezchybná, rozšiřitelná a prakticky neviditelná.

Jak lze poznat špatně navržený web:

1. Každá úprava bude trvat dlouho a klient za ni tedy zaplatí výrazně více, než u dobře navrženého systému.

2. Web bude po každém zásahu někde rozbitý a bude potřebovat manuálně kontrolovat. Dobře navržený web má automatické testy, které programátora upozorní, že něco není v pořádku.

3. Web bude pomalý.

4. Když klient nakonec ztratí trpělivost a předá systém někomu jinému. Následně zjistí, že se v něm nikdo kromě tvůrce nevyzná, nemá přístup ke zdrojovému kódu a dokonce ani nemá právo ho upravovat.

Kombinace nepochopitelných chyb a pomalého webu odradí po určitém čase kteréhokoliv dlouhodobého návštěvníka a sníží šanci získat nové. Rychlost je na webu stále zcela zásadní faktor – neznamená to, že web nemůže např. obsahovat obří fotografie, ale musí na ně být připraven (např. postupným dočítáním stránky[12], které je pro uživatele jasně pochopitelné).

[11] CMS je redakční systém

[12] Lazy loading

Návštěvník očekává, že web bude okamžitě reagovat na jeho akce, protože není ochoten čekat. Lidé s rychlou pevnou linkou nemají naprosto žádnou trpělivost (trpělivost lidí s mobilem není o mnoho vyšší). Rychlost a „performance“ webu je tedy nutné monitorovat a optimalizovat.

Nepodceňujte význam svého vývojářského týmu a rychlosti webu. Bez kvalitního backendu váš projekt z dlouhodobého hlediska ztroskotá.

Rychlost webu je návštěvníky vnímána subjektivně a web lze z designerského hlediska „urychlit“ tím, že člověk vidí, že se něco děje (animujete otevření článku, zatímco ho načítáte na pozadí, v případě delších prodlev zobrazíte indikátor průběhu akce[13]...) – 10 sekund načítání webu bez vidiny pokroku (tedy bez zpětné vazby či jiného podnětu) je věčnost. 10 sekund, kdy se návštěvník baví nebo alespoň vidí, že se něco děje, nemusí být žádný problém. Rychlost je dokonce možné předstírat – např. síť Twitter odeslaný tweet okamžitě zobrazí ve streamu uživatele, ačkoliv ho zatím odesílá na pozadí stránky na server. Ale člověk to nepozná. [5]

3.2.7.4 Přístupnost

Přístupnost se zkomplikovala nástupem vysoce variabilních mobilních zařízení a opět ji musíte vyřešit předtím, než se vrhnete do vyšších pater pyramidy. Přístupnost by měla být primárně doménou webového kodéra[14], který by měl v rámci tvorby webu hlídat, zda se grafický návrh příliš neodklání od zažitých pravidel přístupnosti a dostatečně respektuje variabilitu zařízení. Jako vodítko vám mohou pomoci Pravidla přístupnosti pro weby české státní správy, či některá ze zahraničních metodik, ale ve skrze je jejich dodržování jednoduché – pište hezký HTML kód, mějte na webu jasně odlišená místa, kam lze kliknout (dotknout se,...) a dodržujte kontrast.

Smyslem přístupnosti je zmírňovat či odstraňovat bariéry na webu tak, aby kladl uživatelům co nejméně překážek, a ti jej mohli bez problémů používat a udělat na webu to, proč na něj přišli (přečíst si článek, koupit si zboží či poslat e-mail).

Základní principy přístupnosti

[13] Progress bar

[14] Přístupnost je samozřejmě klíčová i pro grafické a interakční designery.

- Obsah webové stránky je vhodně strukturován pomocí nadpisů.[15]
- Web má dostatečný barevný kontrast písma.[16]
- Text webu je dobře[17] čitelný.[18]
- Web je plně ovladatelný z klávesnice.
- Grafické prvky mají definovanou textovou alternativu.
- Tabulky splňují následující vlastnosti:
  - buňky obsahují pouze informace, které spolu logicky souvisejí,
  - tabulka dává smysl čtena po řádcích,
  - složitější tabulka se sloučenými buňkami či vícenásobným záhlaví obsahuje atribut summary a její buňky jsou svázány s jednotlivými záhlavími.
- Každé formulářové pole má vhodný popisek.[19]

Přístupnost řešíte především kvůli třem skupinám lidí, kteří budou chodit na váš web. Jsou to nevidomí a handicapovaní, roboti vyhledávačů a návštěvníci s mobilními zařízeními.

3.2.7.5 Nevidomí a handicapovaní

Přístupnost vznikla kvůli lidem se zdravotním postižením (nevidomí, tělesně či sluchově handicapovaní). Se svým handicapem se většinou vyrovnávají nástroji a asistivními technologiemi [20], které jim pomohou překonat většinu případných překážek, kterou jim HTML kodér může na webu zanechat.

Handicapovaným návštěvníkům je vhodné se věnovat i v případě, že nejsou členy vaší cílové skupiny – představte si, jaké by to bylo, kdybyste byli na jejich místě. Kromě toho asistivní technologie nejsou všemocné – pokud nefunguje nějaký web v mobilu, můžeme si jej později prohlédnout na desktopu. Ale nevidomý uživatel analogickou možnost (tj. vrátit si zrak a prohlédnout si web očima v případě, že si s ním neporadí pomocí screen readeru)

[15] http://naostri.se/nadpisy

[16] http://naostri.se/contrast-ratio

[17] http://naostri.se/lide11

[18] http://naostri.se/repxpt

[19] http://naostri.se/plavacekform

[20] Screen readery (čtečky/odečítače obrazovky), softwarové lupy, hmatové zobrazovače, speciální klávesnice, hlasové ovládání, atp.

nemá. Handicapovaní nejsou pouze nevidomí – patří sem také lidé s poruchami vnímání barev a dalšími smyslovými poruchami.

V neposlední řadě je vhodné vědět, že úpravy, které se kvůli přístupnosti udelají, nepomáhají jen skupině uživatelů se zdravotním handicapem, ale jejich dopad je mnohem větší [21]. Například nadefinované klávesové zkratky pomohou každému, kdo chce web efektivněji ovládat z klávesnice, dostatečný barevný kontrast zlepší čitelnost textu i pro běžné návštěvníky a přístupně vytvořené formuláře se snáze ovládají nejen z klávesnice, ale i pomocí myši.

3.2.7.6 Roboti vyhledávačů

Druhou kategorií návštěvníků, pro které řešíte přístupnost, nejsou lidé, ale roboti vyhledávačů. Ti vidí váš web primárně jako textovou stránku (Google penalizuje weby, které mají nahoře na stránce bannery [22], takže ji nevidí jenom jako text). Roboti vyhledávačů potřebují přehledně napsaný HTML kód (well-formed) a obsah s textovou alternativou. Pro roboty je rovněž důležitý soubor robots.txt, kterým můžete vyloučit stránky z vyhledávačů, a mapa celého webu sitemap.xml, která zjednodušuje robotům orientaci na stránkách.

3.2.7.7 Návštěvníci s mobilními zařízeními

Třetí a největší kategorii tvoří návštěvníci s mobilním zařízením – telefonem či tabletem [23]. Jejich počet roste a bude pravděpodobně růst nadále na úkor desktopů. Tito návštěvníci vyžadují především:

- layout optimalizovaný pro jejich zařízení,
- dostatečně velké odkazy, na které lze kliknout prstem,
- přizpůsobení obsahu webu vzhledem k jejich aktuálnímu kontextu (hledám na webu adresu kanceláře a mobilní verze webu mi hned na homepage nabídne mapu s vyznačenou cestou ke kanceláři).

[21] http://naostri.se/pristupnost-charita

[22] http://naostri.se/gpenalty

[23] nebo dokonce chytrými brýlemi či hodinkami

Jestliže nemáte o existenci „mobilního webu“ tušení, rychle si zaplňte mezery – návrh webů s vysokou variabilitou zařízení je budoucnost webdesignu. Bez ohledu na nástup displayů s vyšším rozlišením budou mít mobily a další zařízení screeny ergonomicky přizpůsobené lidem. Bude pro ně tedy potřeba vytvářet responzivní či zcela samostatný layout. Pro rychlý vstup na pole mobilního webu uvádím vybrané poučky, které jsem sebíral z různých zdrojů, především od Luka Wroblewskiho: [5]

1. Web musí být správně uspořádán
    - Vždy je zobrazen nejdříve obsah, pak teprve navigace
    - Součástí hlavičky webu jsou odskoky na navigaci/přihlášení/vyhledávání
    - V rámci navigace je odskok zpět (nahoru) na stránku
    - V patičce je přepínač na standartní zobrazení webu (existují uživatelé, kterým mobilní verze prostě nesedne)
2. Web musí být rychlý [24]
    - Neobsahuje přesměrování stránek
    - Používá CSS sprites [25]
    - Kaskádové styly/javascript jsou v jednom souboru a jejich kód je minifikován
    - Postupné nahrávání obsahu
    - Canvas místo obrázků
3. Web musí mít výrazné a dostatečně velké aktivní prvky
    - Prst klikne na oblast cca $9 \cdot 9$ mm
    - Důležitá tlačítka a tlačítka blízko okraje displaye jsou větší
    - Web musí mít mezi aktivními prvky mezery
    - Okolo aktivních prvků je neviditelná oblast, na kterou lze kliknout
4. Web je nezávislý na stavech po najetí myší
    - Formuláře jsou přizpůsobeny chování mobilních prohlížečů
    - Popisky formulářů musí být umístěny nad příslušnými poli, nikoliv vedle nich – mobilní telefony často formuláře přiblíží a popisky vedle polí nejsou vidět
    - Pole formulářů jsou sémanticky označena – uživateli se tedy zobrazí při zadávání e-mailu upravená klávesnice pro e-mail

[24] Rychlost výrazně ovlivňuje použitelnost webů na mobilech – pokud je web pomalý, obecně to zhoršuje jejich vztah k brandu provozovatele. http://naostri.se/slowdmg

[25] Více obrázků v rámci jednoho velkého obrázku, které pozicujete pomocí CSS

3.2.7.8 Použitelnost

Použitelný web nehází návštěvníkovi klacky pod nohy. Návštěvník se v něm zorientuje a je schopen provést rychle to, proč na web přišel – ví, kde je, kam může jít a na co lze kliknout. Použitelnost můžete měřit rychlostí provedených akcí a množstvím chyb, které návštěvník udělá při používání webu. Součástí použitelnosti je i snadnost konzumace obsahu návštěvníkem – použitelný web je pro návštěvníka srozumitelný a návodný.

Vyšší metu použitelnosti tvoří intuitivnost – tzn. web odpovídá návštěvníkovým fyzickým a mentálním predispozicím. Rozdíl mezi použitelností a intuitivností je přibližně takový, že mapu můžete ovládat pomocí růžice (což je použitelné a lidé se rychle naučí klikat na šipky, aby se mapa pohnula) nebo tahem myši/prstu (což je intuitivní).

Z hardwarového pohledu je dnes standard používat pro ovládání počítače myš (a lze se to poměrně rychle naučit), ale intuitivní je používat prsty a přímé dotyky na obrazovce.

Součástí použitelnosti je i praktická aplikace typografie a teoretických principů, které umožní návštěvníkovi v lepší orientaci na webu, a aplikace návrhových vzorů, které návštěvníkovi zjednodušují používání webu. K použitelnosti patří také dobře navržená informační architektura webu a vhodně zvolené priority obsahu při návrhu jednotlivých stran.

O použitelnosti je toho napsáno mnoho – je to základ, od kterého se potřebujete při návrhu webu odrazit (přístupnost a dostupnost za vás z velké části vyřeší HTML kodér a programátoři). Nedá se přeskočit, nedá se obelstít a má jednoznačný dopad na obchodní úspěch webu. Stejně tak není nijak složitá, pár stovek pravidel a použitelný web je na světě. Možná proto je na některých webech vidět, že použitelnost už přestala webdesignera bavit a snaží se o rádoby inovativní přístup. Inovace bez použitelnosti nebude nikdy fungovat. Nepokoušejte se inovovat uživatelské rozhraní dříve, než pochopíte principy, na kterých stojí vnímání lidí, a tudíž i použitelnost. [5]

3.2.7.9 Pocit bezpečí

Důvěryhodnost webu je vhodné v některých momentech umocnit tím, že vizuálně podpoříte pocit bezpečí. Typicky při placení online je vhodné těsně vedle formuláře, kam člověk zadává

číslo platební karty, umístit informaci, že je to bezpečné [26], vše je zaštítěno důvěryhodnou autoritou, spojení je šifrované SSL (a skutečně ho mít šifrované),...

Web musí odpovídat na dotaz svých návštěvníků – co se stane s číslem mé kreditní karty? Je tato transakce bezpečná? Komu prodáte můj e-mail? Bez odpovědí na tyto otázky je často zmařeno množství konverzí, které by jinak proběhly. [5]

3.2.7.10 Kdo je webdesigner?

Webdesigner v kontextu této práce není ve většině případů jeden člověk, ale celý tým lidí. Jako webdesigner (tedy to, co si lidé běžně pod pojmem webdesigner představují) lze dělat jednu z následujících činností nebo jejich kombinaci:

- návrh webu (výstupem je dokumentace k webu) → UX designer, interakční designer, informační architekt,...
- grafika (výstupem jsou obrázky webů) → webový grafik
- HTML (výstupem webová stránka) → webový kodér
- HTML + javascript (opět je výstupem webová stránka) → front-end developer

Z hlediska náročnosti je nejjednodušší začít jako kodér nebo webový grafik, protože v takovém případě převládají řemeslné aspekty webdesignu. V případě front-end developera přidejte nutnost programovat, jako UX designer potřebujete více interakčního designu, informační architektury, strategie, komunikace,...

Moje pojetí webdesignera je kombinací povolání výše + přehled o SEO, copywritingu, marketingu, brandingu, psychologii, gamifikaci, byznysu, atd. - což je ovšem stav, do kterého se nedostanete za rok. Webdesigner je desetibojař – nic nedělá do hloubky, ale je všestranný, aby se mohl rychle přizpůsobovat vývoji našeho oboru. Každé z prezentovaných povolání má své prerekvizity, klady a výzvy a potřebujete u něj dosáhnout určité množství teorie. [27] [3, 5, 14]

[26] http://naostri.se/checksec

[27] V době psaní této práce vznikají v rámci Sektorové rady pro IT pod MPO a MŠMT hodnotící standardy pro tři z výše uvedených povolání – interakční designer, webový grafik a webový kodér. V praxi to znamená, že existuje seznam zcela konkrétních znalostí a dovedností pro tato povolání a je posvěcen výše zmíněnými institucemi.

3.2.7.11 UX designer

Úkolem UX designera je transformovat myšlenky klienta a pochopit jeho zákazníky tak, aby mohl navrhnout prototyp webu a web průběžně testovat. Díky pochopení vstupů a jejich transformaci na výstupy má vizi konečného výsledku a stará se, aby byla realizována napříč celým projektovým týmem tak, aby bylo dané vize dosaženo. Jeho role je být ve středu každého projektu – projektový manažer se stará o hladký průběh a technické vedení projektu a UX designer mentálně propojuje všechny účastníky projektu. Když někdo v týmu potřebuje ujasnit některou z částí webu, nejde za projektovým manažerem, ale UX designerem.

UX designer vytváří prototyp a dokumentace popisující části webu, které nedokáže popsat prototypem. K tomu potřebuje znalosti uživatelského výzkumu, uživatelského testování, interakčního designu, informační architektury, psychologie a strategie obsahu (čím větší weby navrhujete, tím více a lépe). UX designer musí přemýšlet a dávat web do souvislostí – potřebuje hluboké teoretické základy. Do řemeslné části pak patří tvorba dokumentace popisující web (tak, aby vypadala dobře a byla jednoduše konzumovatelná) a prototypů – ať už v grafickém editoru, nebo některém z programů, které jsou pro prototypování webů určeny. Jestliže chce UX designer navrhovat i rozložení prvků na stránkách prototypu, pak musí mít zkušenost s grafickým designem. UX designer aplikuje své znalosti psychologie, sociologie, marketingu, brandingu,... třeba v rámci procesu popsaného v první části práce, aby navrhl funkční web. Wireframy jsou pro něj jeden z prostředků pro předání idejí, nikoliv modla, okolo které se vše točí. [3, 5, 14]

Typická práce UX designera:

- Zkoumá kontext ovlivňující projekt na straně klienta i jeho zákazníků.
- Plánuje a provádí jednotlivé části procesu návrhu webu.
- Provádí uživatelský výzkum.
- Pomáhá vymezit rozsah projektu.
- Pomáhá pojmenovat problém a hypotézy pro jeho řešení.
- Volí psychologické principy aplikované na konkrétním webu.
- Volí metody a způsoby dokumentace.
- Testuje průběžně hotová řešení.

- Nasazuje a vyhodnocuje analytické nástroje.
- Dohlíží na celý proces tvorby webu.

3.2.7.12 Webový grafik

Webový grafik vytváří náhledy webových stránek v grafickém programu a potřebuje grafický cit. Jeho práce tedy stojí zejména na tradičním grafickém designu, typografii a na souvisejících teoretických znalostech (např. teorie gestaltu). Webový grafik si nemusí lámat hlavu s návrhem obsahu, ale musí skvěle graficky interpretovat wireframy, aby měl výsledek smysl pro koncového uživatele a zapadal do vizuálního stylu klienta. Tzn. nejde pouze o obarvování polí, ale o přizpůsobování wireframů s ohledem na jejich skutečný obsah a vizuální prvky – to, co bylo ve wireframu bílý proškrtnutý čtvereček, může být v grafice cokoliv.

Řemeslnou částí práce webového grafika je ovládnutí grafického programu – standard pro webového grafika je v roce 2017 Adobe Photoshop. Na grafické programy existuje množství postupů a návodů na internetu (a samozřejmě knih), můžete s tím začít prakticky okamžitě. Řemeslná schopnost interpretovat wireframy, cit pro leštění barevných ploch a grafický design z vás ale ještě nedělají webdesignera. Jste vždy závislí na kvalitě výstupů UX designera. Je proto potřeba před vstupem na volnou nohu doplnit minimálně teorii a praxi interakčního designu. [3, 5, 14]

Typická práce webového grafika:

- Vytváří moodboard webu.
- Pojmenovává emoce, které má web mít.
- Zasazuje web do grafického stylu klienta.
- Koordinuje volbu a tvorbu grafických podkladů – ilustrací, videí či fotografií, které budou součástí webu (nemusí je nutně vytvářet – webový grafik nemusí být ani ilustrátor, ani DTP operátor, ale obojí se samozřejmě hodí).
- Navrhuje typografickou mřížku a konzistentní typografii celého webu.
- Graficky interpretuje wireframy.
- Vytváří vizuální framework pro tvorbu dalších prvků webu.

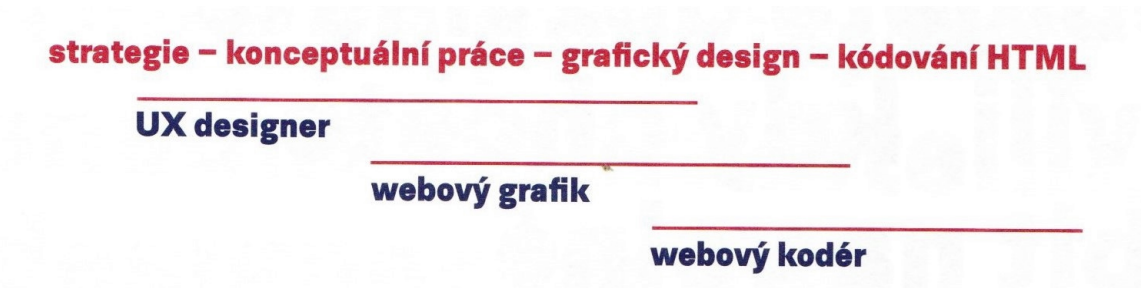
3.2.7.13 Kodér / front-end developer

Kodéřina má nejblíže k řemeslu – dostanete webovou stránku jako obrázek a přepíšete ji do HTML+CSS tak, aby se web zobrazoval správně v jednotlivých prohlížečích/zařízeních/operačních systémech. HTML+CSS se lze naučit rychle, takže je to nejjednodušší vstupní brána do webdesignu. Výrazně složitější je pak přizpůsobování výstupů webového grafika vzhledem k rozlišení prohlížeče, tvorba responzivních webů a především udržování webu v konzistentním a snadno rozšiřitelném stavu. Webový kodér bude mít bez typografického citu velice těžkou práci, protože s ním bude muset grafik ladit každý pixel (nebo desítku pixelů – záleží na přístupu grafika – tak jako tak ho to brzy přestane bavit). [3, 5, 14]

Typická práce webového kodéra:

- Udržuje konzistenci grafických prvků napříč webem.
- Sjednocuje designové výstřelky grafika.
- Zasazuje elementy webu do mřížky.
- Vytváří systém pro udržování kódu, aby bylo web možné jednoduše rozšiřovat.
- Důsledně udržuje vizuální framework webu.
- Vytváří responzivní varianty layoutu pro mobilní telefony a tablety.

Front-end developer si k HTML musí připočíst javascript, a tudíž znalosti týkající se programování. [5]



Obrázek 11: strategie – konceltuální práce – grafikcý design – kódování HTML [5]

4 Vlastní práce

4.1 Web design

„Uděláme si nyní krátký rychlý výlet do historie. Před 10 lety to ve webdesignu bylo krásně jednotné. Napsala se analýza. Tedy pardon, analýzy nikdo nepsal. Udělala se rovnou nabídka. Když tam byla analýza, pak například tato – 'cílová skupina webu jsou muži a ženy mezi 18 a 70 lety'. Ve Photoshopu se navrhla grafika. Klient ji pak schválil nebo se s klientem iterovalo až do kompromisního výsledku. Nebo neskutečné slátaniny. Grafika se převáděla do HTML a CSS. Nikdo nechápal, k čemu je frontendista dobrý. 'Vždyť už to nakreslil grafik' říkali klienti. Šablony se programovaly. Programátor si nasadil sluchátka a 3 měsíce programoval. Sluchátka si sundal a ukázal výsledek. Všichni se zhrozili, udělal úplně něco jiného. Web se spustil. Oproti odhadům z analýzy – ach pardon nabídky! – trvala výroba webu třikrát déle. Ten poslední s nadějí, že to dopadne dobře, to tady vzdal. Ano, tenhle proces je z pekla. Říkává se mu vodopád.“ Takto uváděl na WebExpu 2015 svoji přednášku Adam Kudrna. [1, 4] Na základě toho upravil i proces tvorby uživatelského rozhraní, který se snažím v této práci použít.

Na začátku je tedy analýza a uživatelský průzkum. Na to navazuje tvorba wireframů a až na ně navazuje grafika. Zbytek procesu zůstává podobný. Obohacený je ještě o vyhodnocení a po něm opět návrat na začátek další iterace. Budeme se však zaměřovat na část návrhu a realizace uživatelského rozhraní.

4.2 Prototypování

Prototypování jako pojetí existovalo už před Internetem. Lidé, kteří vyvíjeli nějaký nový fyzický produkt, obvykle nejprve tu věc sestavili a přesvědčovali se, zda funguje tak, jak si to představovali. První verzi si pak nechali patentovat a ukázali ji potenciálním investorům. Pokud měli vynálezci přístup k jistým výrobním prostředkům (například pracovali v nějaké továrně), pokračovali ve vývoji dál, odstraňovali chyby a nedostatky, dokud se nedostali až k modelu, který se mohl rovnou začít vyrábět. [10]

Prototypy se hodně podobají starým demům, jako je například staré demo CD, na nichž byly programy s omezenou funkcionalitou, nebo hry jen s jednou nebo dvěma herními

úrovněmi, jen jsou ještě jednodušší.

Jejich účelem není v první řadě přesvědčit perspektivní zákazníky, aby produkt zakoupili, spíše má napomoci udělat ho lepší. Prototyp webu nebo aplikace sestavujeme proto, abychom se přesvědčili, že bude fungovat tak, jak jsme zamýšleli. Můžeme s ním také klientům nebo potenciálním investorům ukázat, jak předpokládáme, že bude fungovat.

4.2.1 Wireframy a mockupy

Každý návrhář, který se už nějakou dobu pohybuje v profesi, se už patrně setkal či pracoval s wireframy (tzv. skici webu) a (nebo) v nějakém obrázkovém editoru navrhoval mockupy (někdy se jim říká statické nákresy webu).

Wireframy (skici) se dají vyrábět na papíru nebo v aplikacích. Téměř vždy se považují za prototypy s nízkou precizností, přestože je můžeme povýšit na prototypy se střední precizností, pokud do nich investujeme patřičně času. Vynaložené úsilí se však vyplatí jen výjimečně.

Wireframy se navrhují obvykle tak, aby se daly rychle kreslit i zahazovat. Právě proto je tak atraktivní kreslit je na papíru. Aplikace sice mohou být daleko preciznější a existující wireframy také můžeme snadno upravovat; nic ale nepřekoná rychlost, s jakou jsme schopni udělat si náčrtek, který je určen jen nám, nikomu jinému … nikdy.

Wireframy založené na aplikacích mají tu přednost, že zase snadněji napodobují funkcionalitu rozhraní. Když ťukáme na papírový náčrtek, slyšíme jen plesk, plesk, plesk, nic se neděje.

Může to být zábavné, ale možná to nevyjadřuje, co jsme tím chtěli sdělit.

Můžeme však zvolit kombinaci obojího: na papíru podchytit nejzákladnější pojmy, a pomocí aplikace je dotvářet a snadno sdílet. [10]

4.2.2 Prototypování v prohlížeči

Hlavní potíž s wireframy nebo mockupy je v tom, že jsou statické. Existuje značná spousta informací týkajících se jejich funkcionality, které jednoduše nedokážou vyjádřit. To může vést k mylným představám a nedorozuměním v myslích klientů nebo dokonce vývojářů o tom, jak se předpokládá, že bude daná věc fungovat. Z tohoto důvodu bývají prototypy obvykle nějakým způsobem interaktivní.

Problémem je, že s prohlížečem a reálným zařízením – skutečným prostředím pro svůj běh – se návrh potká až pod rukama frontendisty. Do prohlížeče se návrh dostane pozdě. A obvykle až v prohlížeči se vynoří řada problémů souvisejících hlavně s responzivním designem nebo třeba rychlostí načítání, které designér ani grafik vidět nemohl. Používají totiž statické nástroje, které webová prostředí maximálně napodobují. Špatné je, že na vyřešení těchto problémů už pak není čas. Proto je zlé, že s prohlížečem se rozhraní potká takhle pozdě. V některých případech tento proces produkuje nekvalitní výsledky, v jiných zase zbytečně spotřebovává čas a peníze. Chceme tedy mít web v prohlížeči co nejdříve. Takže spojíme designovací a kódovací fázi do jedné – designování v prohlížeči. Prohlížeč vede za ruku i nezkušeného designéra. Z obrázků to vidět není, ale když držíte lineární prototyp v mobilu, designérské problémy na vás vyskakují samy. Učíme se designovat palcem na displeji mobilu. Jsme odstínění od dalších vrstev problémů jako je design jednotlivých komponent, celostránkový design nebo grafika. Myslím, že tímhle postupem se i méně zkušení webaři mohou naučit rozumně navrhovat uživatelská rozhraní. Dynamičnost webu je v interakcích, animacích a v našem případě hlavně responzivitě – přizpůsobování rozhraní různým rozlišením. [1]

Web = multimediální hypertext + interakce + responzivnost

Ano, můžeme si udělat responzivní prototypy třeba v Axure, i s exportem do HTML. To, co vznikne, ale nepoužívá plnohodnotnou HTML/CSS technologii. Responzivní web tam pouze emulujete. Fluidní layout – přizpůsobování rozhraní pixel po pixelu tam neuděláme. Je to něco mezi statickým návrhem a webem v prohlížeči.

Jedině přímo prohlížeč a testování na reálných zařízeních vám řekne, jak budou prvky ovládatelné palcem, na velikosti displeje zařízení X, v prohlížeči Y, vykreslené fontem Z. Nebo na pomalém připojení; nebo když uživatel použije zoom; nebo když havaruje Javascript; nebo když… Sami víte, že variant prostředí běhu uživatelského rozhraní je na webu neskutečné množství.

Axure vám neřekne, zda je návrh technicky realizovatelný bez nějakých složitostí typu změny pořadí v kódu nebo detekce na serveru. V prohlížeči si to rovnou můžete zkusit. U složitých komponent se vždy rozhodujte v prohlížeči. Vůbec to neznamená, že musíme zahodit Axure a Photoshop a vše dělat v prohlížeči. Ne každý kodér chce být zároveň designérem a naopak. Ale pokud u webu navrhujeme složité nebo nestandardní UI komponenty, rychlý

prototyp v prohlížeči a vyzkoušení na zařízeních opravdu pomůže.

Designování celých webů v prohlížeči je poněkud extrém. Pojďme se bavit o přenesení důležitých rozhodnutí do prohlížeče. Nemusíme v prohlížeči vytvářet návrh celého webu, stačí se v něm rozhodovat o netriviálních částech. Důležité je, aby to proběhlo už v rané fázi návrhu.

4.2.3 CSS Frameworky

CSS frameworky jsou předpřipravené softwarové frameworky, které jsou určeny k usnadnění tvorby webu pomocí kaskádových stylů. Většina takovýchto frameworků obsahuje alespoň nějaký systém mřížky. Dále řeší třeba reset stylopisu, responzivitu, typografii, ikony, tlačítka a jiné grafické ovládací prvky. Pro tuto práci jsou prakticky použitelné pouze dva hlavní CSS frameworky, a to Bootstrap a Foundation. [18]

4.2.3.1 Bootstrap [28]

Bootstrap je jednoduchá a volně stažitelná sada nástrojů pro tvorbu webu a webových aplikací. Obsahuje návrhářské šablony založené na HTML a CSS, sloužící pro úpravu typografie, formulářů, tlačítek, navigace a dalších komponent rozhraní, stejně jako další volitelná rozšíření JavaScriptu. Pro použití Bootstrapu jsou nutné základní znalosti HTML a CSS, interaktivní prvky jako jsou tlačítka, boxy, menu a další kompletně nastavené a graficky zpracované elementy je totiž možné vložit pouze pomocí HTML a CSS.

4.2.3.2 Foundation [29]

Foundation je responzivní frontendový framework. Poskytuje responzivní mřížku a HMTL a CSS UI komponenty, šablony, znovupoužitelné kusy kódu, včetně typografie, formulářů, tlačítek, navigace a dalších komponent rozhraní, stejně tak jako javascriptová rozšíření. Foundation je spravován společností ZURB a jedná se o open source projekt.

[28] http://getbootstrap.com/

[29] http://foundation.zurb.com/

4.3 Optimalizace rychlosti

Proč zrychlovat webové stránky? Zdlouhavé načítání webu může v mnoha případech odradit návštěvníky. I když je v dnešní době internetové připojení velice rychlé, stále je možné najít uživatele internetu s pomalejším připojením nebo s mobilním internetem.

Je potřeba také myslet na to, že i vyhledávače sledují rychlost načítání webových stránek a v případě jejich zdlouhavého načítání může dojít k určitým postihům (více informací naleznete v článku Using site speed in web search ranking [30] na Webmaster central blog [31]).

Jaké jsou možnosti?

Webové stránky nebo aplikace je možné zrychlit několika způsoby:

- Minimalizace HTTP požadavků,
- Snížení velikosti externích souborů,
- Snížení objemu dat stahovaných ze serveru.
- Samozřejmě je také možné webovou aplikaci zrychlit optimalizací samotného zdrojového kódu webu.

4.3.1 Měření rychlosti webů

Každý dnes očekává, že když zadá ve svém prohlížeči URL adresu nebo klikne na odkaz, okamžitě se mu načte požadovaný web. A Google se pro svůj vyhledávací algoritmus, čítající několik set proměnných, snaží co nejvíce inspirovat samotnými uživateli. Proto je logické, že jedním z parametrů, které ovlivňují výsledné pozice na SERP (výsledcích vyhledávání) je i rychlost načítání webových stránek.

Google oficiálně zahrnul rychlost webu do vyhledávacího algoritmu už v dubnu 2010 [32]. Vycházel tehdy z několika svých interních studií, které prokázaly, že na pomalém webu uživatelé stráví mnohem méně času.

Rychlost webu by měla být pro jakéhokoliv provozovatele prioritou už z toho důvodu, že pokud se uživateli daná stránka nenačte do tří sekund, má to negativní konsekvence. Po tomto časovém limitu většina lidí totiž ztrácí zájem o stránku a odchází. A nemusí se jednat jen o nezájem v okamžik, kdy web dočasně nefunguje či reaguje pomalu. Ne nadarmo se říká,

[30] https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking.html
[31] https://webmasters.googleblog.com/
[32] https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking

že čas jsou peníze. V online prostředí to platí dvojnásob. Negativní zkušenost s načítáním stránek může znamenat i trvalou nedůvěru uživatelů k webu, značce a podnikání!

Existuje celá řada volně dostupných nástrojů, pomocí nichž dokážeme velmi snadno změřit rychlost načítání webu a odhalit i případné nedostatky a riziková místa, která je nutné opravit. [20, 9, 15, 13]

4.3.1.1 Google PageSpeed Insights [33]

Validátor základních technických problémů, které komplikují rychlost webu. Zde by měl každý začínat. Otestujeme si tady všechny důležité vstupní šablony.

PageSpeed Insights (PSI) se dá instalovat i jako rozšíření do prohlížečů nebo testování zautomatizovat pomocí API.

Dokud PSI nedosahuje skóre přibližně kolem 80 bodů na desktopu i mobilu, nemá smysl se učit další nástroje.

4.3.1.2 WebPagetest.org [34]

Dělá pokročilou analýzu, testuje jinak než Page Speed Insights. Na druhou stranu – WebPagetest nelze používat průběžně, testy nějakou dobu trvají. WebPagetest také není tak intuitivní jako PSI, je potřeba mít již hlubší znalosti problematiky.

Umožňuje testování z jiné lokality, testování pomalého připojení a v prohlížečích, ve kterých nemáme pokročilé vývojářské nástroje. Například v mobilních nebo ve starých Internet Explorerech.

Má také API, ve verzi zdarma je omezené na pár stovek dotazů týdně.

4.3.1.3 Chrome DevTools

Pokročilá analýza a detailní testování procesů načítání v Chrome. Velmi podobný nástroj mají i ostatní moderní prohlížeče.

[33] developers.google.com/speed/pagespeed/insights

[34] webpagetest.org

4.3.1.4 GTmetrix [35]

Obsahuje analýzu z Google Page Speed Insights a zároveň ještě YSlow metodiku v jednom reportu.

Umí toho hodně. Ukáže timeline, zvládne emulaci pomalého připojení. Testovací lokality má GTmetrix ale pro ČR horší než WebpageTest.org a s méně možnostmi nastavení. Praktické je monitorování a nastavení připomínek do mailu.

4.3.1.5 Google Analytics

Statistiky z Google Analytics jsou hlavní pomocí pro marketing. Mají ale velmi zajímavé využití i pro vývojáře, hlavně když se rozšíří o Trackomatic a Technical Performace Dashboard.

Je ale potřeba se podívat do Chování → Rychlost → Přehled. Je potřeba měřit pomocí verze Universal Analytics. Analytics ukazují Časování stránek (Page Timings), ale napříč různými kontexty – prohlížeče, regiony.

V Časování uživatelů (User Timings) mohou být vlastní měření – např. jak rychle se načetl konkrétní obrázek. Je potřeba to nastavit.

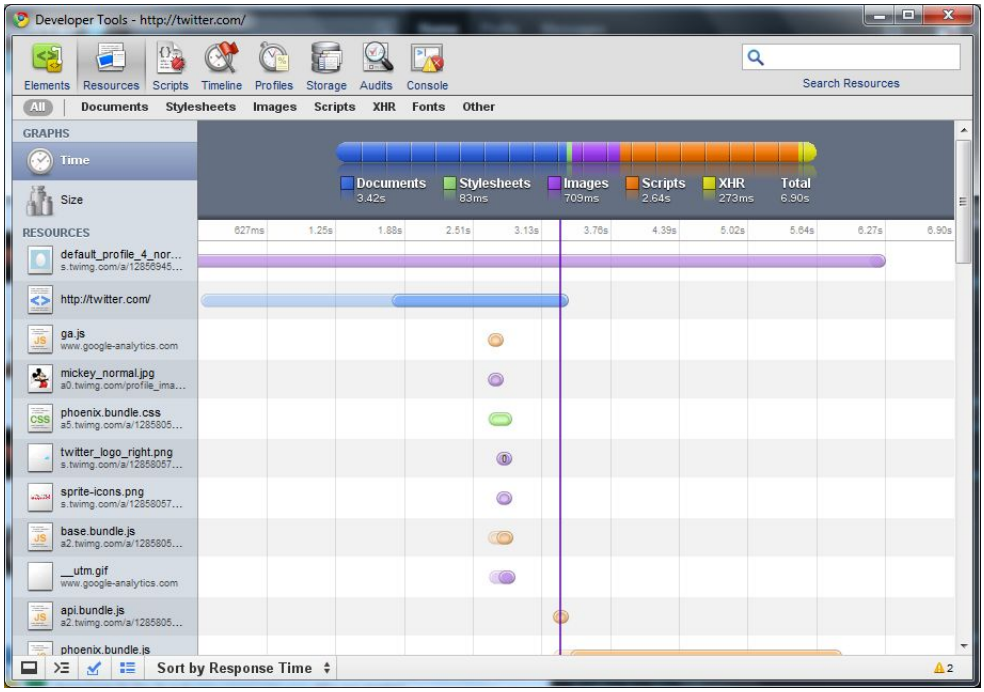
Standardně se pro měření rychlost používá jednoprocentní vzorek shlédnutí měřené stránky.

4.3.2 Minimalizace HTTP požadavků

Jedním z nejefektivnějších způsobů, jak zrychlit načítání webu, je snaha o co nejmenší počet stahovaných dat ze serveru, například:

- **Sjednotit externí soubory** – CSS styly nebo JavaScript je vhodné umístit vždy do jednoho souboru, i když jeho výsledná velikost bude vyšší. Každý stažený soubor musí prohlížeč po jednom zpracovat a následně využít. To může zpomalovat zpracování ostatních načítaných souborů.
- **Využít cachování** – díky cachování (ukládání do mezipaměti) je možné opravdu efektivně minimalizovat počet všech načítaných externích souborů.

[35] gtmetrix.com



Obrázek 12: Stahované externí soubory na Twitter.com (zobrazeno v Developer Tools prohlížeče Google Chrome) [9]

4.3.2.1 Kešování

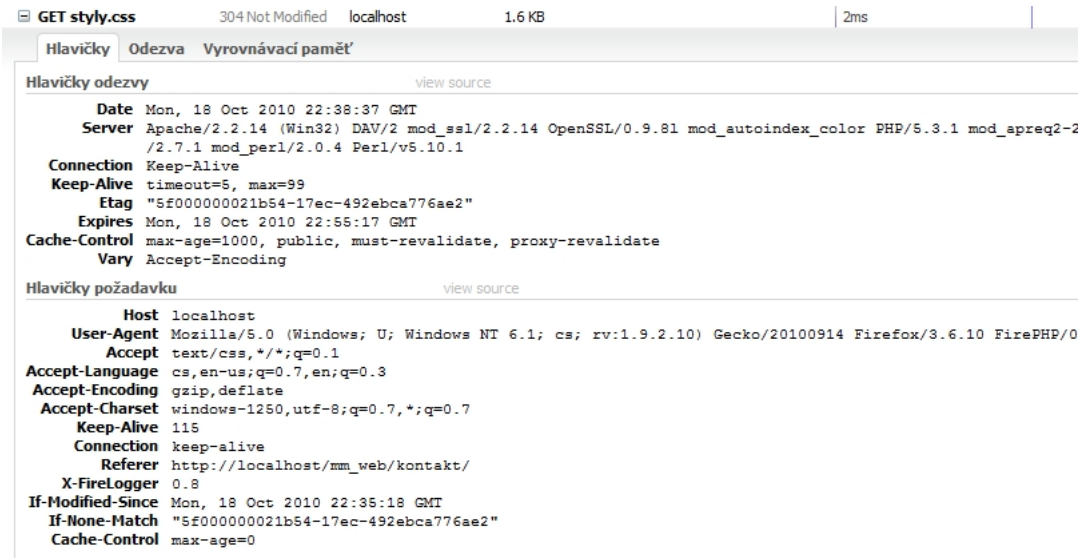
Často si můžete povšimnout, že se při opakované návštěvě web načetl rychleji než v případě první návštěvy.

Data se při první návštěvě uložila do mezipaměti prohlížeče, tzv. cache, a při opětovné návštěvě se nemusela ze serveru opět stahovat (neuložený nebo aktualizovaný obsah se opět stáhl ze serveru). Díky tomu je možné snížit počet HTTP požadavků, ale také snížit celkový objem přenášených dat.

Jak to funguje?

Aby se data správně ukládala a aktualizovala, tzv. invalidovala, je potřeba správně nastavit hlavičky požadavků. Ty se poté využívají při komunikaci pro ověření platnosti dat přes HTTP. Pokud nejsou hlavičky nastavené, uložení dat bude pouze dočasné v rámci sezení. [36]

[36] Záměrně nezmiňuji možnost zpracování dat v mezipaměti pomocí HTML tagů. Metoda není příliš spolehlivá vzhledem k tomu, že hlavičky odeslané serverem mohou význam tagů přepsat.



Obrázek 13: HTTP hlavička požadavku na soubor styly.css [9]

Jak můžete vidět na obrázku, při zasílání požadavků pro stažení souboru ze serveru obsahuje hlavička několik parametrů (Hlavičky požadavků):

- Cache-Control,
- If-None-Match,
- If-Modified-Since.

Ty server vyhodnotí a odešle na ně odpověď (Hlavičky odezvy):

- Expires,
- Etag,
- Cache-Control.

V hlavičce odezvy Cache-Control jsou další parametry, které říkají, jak se souborem v mezipaměti zacházet:

- Max-age – doba platnosti souboru, tzv. čerstvost. Po jejím uplynutí bude vyslán požadavek na jeho stáhnutí.
- Public – data budou „cachována“ v rámci všech cache (i když budou data v zabezpečené části).
- Private – opak parametru public (určíme, že data jsou privátní a nebudou se nijak ukládat).

- No-cache – platnost dat se bude ověřovat vždy při vytvoření požadavku – před samotným stažením souboru se ověří jeho platnost.
- Must-revalidate – pokud potřebujeme soubory ověřovat při každém zobrazení stránky, a vždy při jejím načtení se provedou všechna nastavená ověřovací pravidla.
- Proxy-revalidate – stejné jako must-revalidate, akorát v rámci proxy cache.

Příklad využití parametrů

Pokud potřebujeme ověřit platnost dat uložených v mezipaměti, využijeme k tomu parametry *If-None Match* a *Etag*, který obsahuje hash obsahu souboru a který se využije k porovnání mezi hlavičkami. Tento mechanismus je řízen serverem automaticky.

Hlavička *If-Modified-Since* kontroluje, zda byla data od uvedeného data nějak upravena. Jak ukazuje příklad na obrázku, pokud byl soubor upraven od 18. října 2010 22:35:18, data se invalidují a budou opět stažena.

Pokud bychom chtěli zabránit neustálému ověřování dat v mezipaměti, což samozřejmě také prodlužuje dobu načtení webu, může být vhodné pro jednotlivá data nastavení času, po který se nemusí platnost dat ověřovat. To je možné díky hlavičce Expires. Může však dojít k problémům kvůli špatné synchronizaci času na serveru, a proto je vhodnější využít ke kontrole Etag.

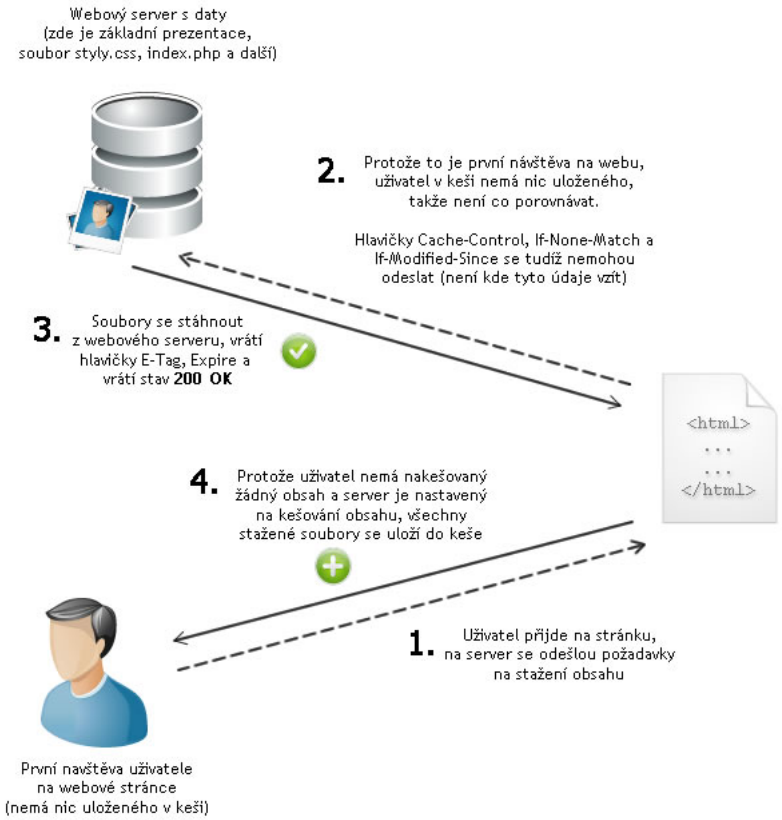
Stavové kódy v hlavičce

Pokud budou všechny hlavičky platné, odpověď pro data bude stav *304 Not Modified* a data se načtou z mezipaměti. Pokud by se ale soubor změnil a některá z hlaviček by se vyhodnotila jako neplatná, odpovědí bude stav *200 OK* a soubor se opět stáhne ze serveru.

Pro lepší pochopení stavových kódu a parametrů v hlavičkách jsem přiložil jednotlivé situace, které mohou při požadavcích vzniknout.

Při první návštěvě nejsou v mezipaměti uložena žádná data, a proto není možné žádnou hlavičku porovnat. Všechny požadované soubory budou staženy a uloží se do mezipaměti pro případ další návštěvy:

Při první návštěvě se data uložila do mezipaměti. Před další návštěvou byl na serveru



Obrázek 14: První návštěva uživatele na webu (mezipaměť neobsahuje žádná data), stav je 200 OK a data se budou stahovat [9]

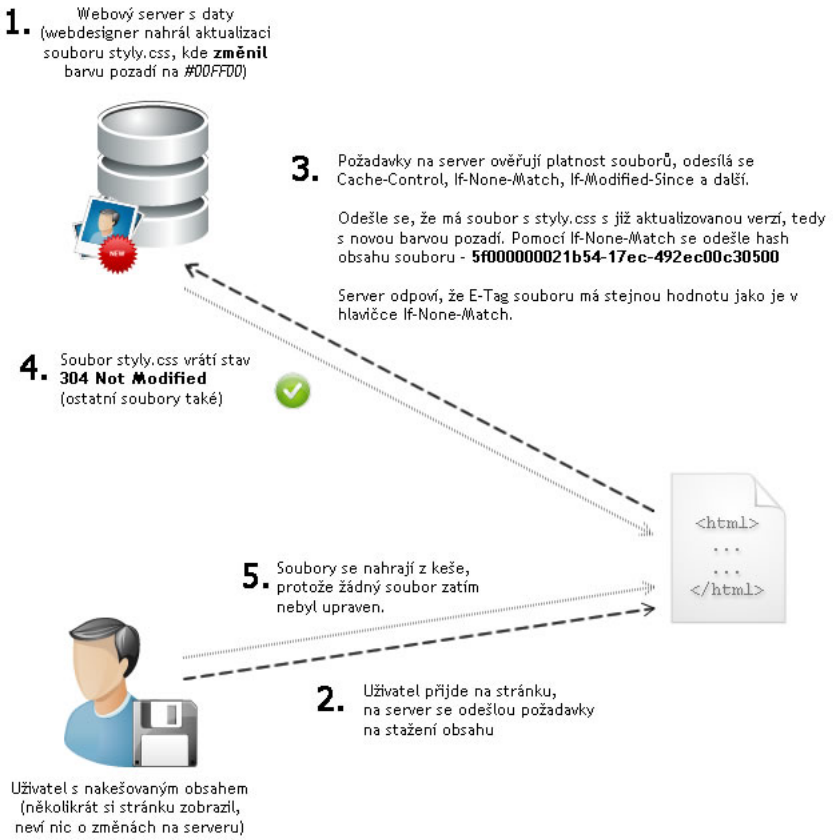
upraven soubor se styly.css (změnil se jeho obsah) a pro který byla nastavena pravidelná kontrola jeho platnosti pomocí must-revalidate.

Při opětovné návštěvě stejného uživatele dojde k vytvoření požadavku na kontrolu data v mezipaměti. Pomocí hlavičky If-None-Match dojde ke zjištění, že soubor styly.css byl změněn, a bude opět stažen (ostatní se nahrají z mezipaměti).

Před další návštěvou webu nedošlo k úpravě žádného souboru. Data v mezipaměti prohlížeče se tedy vyhodnotí pomocí *If-None-Match* a *Etagu* a jako platná se načtou právě z mezipaměti.

Díky tomuto mechanismu lze velmi efektivně snížit čas potřebný pro nahrávání webu a snížení celkového objemu stahovaných dat – stáhnou se vždy až v případě, kdy bude třeba. Tento způsob je velmi vhodné využívat v případě mobilních webů.

Nastavení na webovém serveru



Obrázek 15: Opakovaná návštěva na stejném webu o několik dní později. Během té doby byl na serveru upraven soubor styly.css [9]

Aby vše správně fungovalo a hlavičky obsahovaly potřebné parametry, je potřeba vše nastavit na webovém serveru. Jako příklad uvedu nastavení pro Apache 2 [37] s využitím modulů mod_header [38] a mod_expires [39].

Vše je pak řízeno nastavením v souboru .htaccess [40] umístěném v kořenovém adresáři webu. V něm je dobré zapnout direktivu pro nastavení expirace:

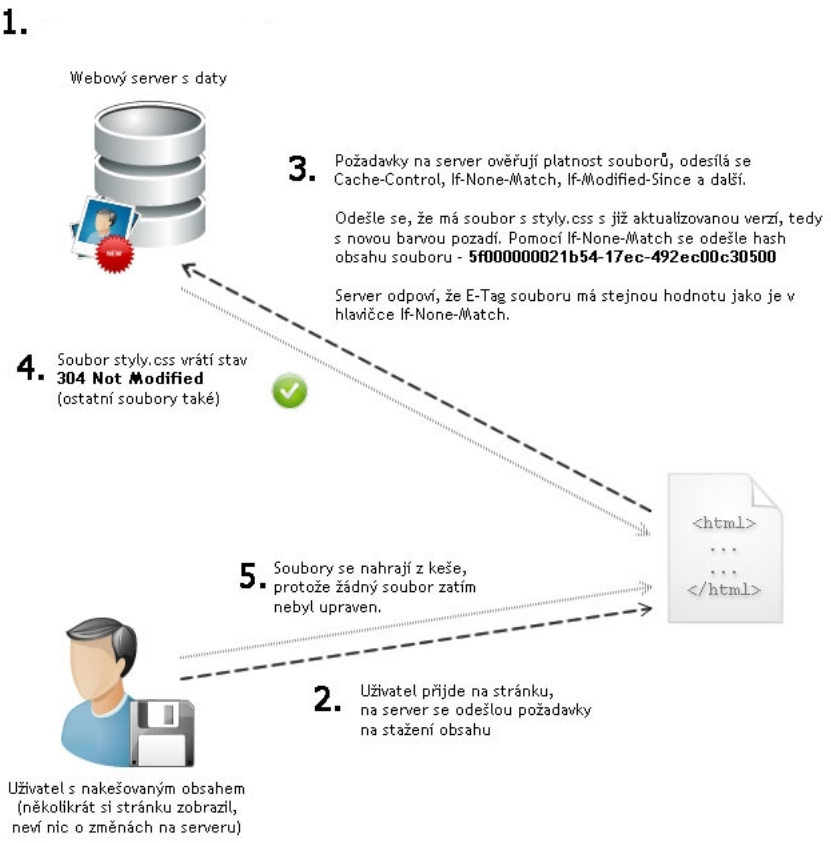
```
<IfModule mod_expires.c>
  ### aktivace modulu mod_expires
  ExpiresActive On
</IfModule>
```

[37] http://httpd.apache.org/

[38] http://httpd.apache.org/docs/current/mod/mod_headers.html

[39] http://httpd.apache.org/docs/current/mod/mod_expires.html

[40] https://www.jakpsatweb.cz/server/htaccess.html



Obrázek 16: Opakovaná návštěva na stejném webu o několik dní později. Žádný ze souborů nebyl na serveru změněn. [9]

Pro cachování jednotlivých souborů se pak použijí pravidla s regulárním výrazem (porovnání podle přípony):

```
<IfModule mod_expires.c>
  ### aktivace modulu mod_expires
  ExpiresActive On

  <FilesMatch "\.(ico|pdf|flv|jpg|jpeg|png|gif|js|css|swf)$">
  Header set Cache-Control "max-age= 2592000, public"
  </FilesMatch>
</IfModule>
```

Tímto zajistíme, že všechna statická data, např. obrázky, javascriptové a CSS soubory budou uložena po dobu 30 dní v mezipaměti.

```
<IfModule mod_expires.c>
  ### aktivace modulu mod_expires
  ExpiresActive On

  <FilesMatch "\.(html|htm)$">
  Header set Cache-Control "max-age=7200, must-revalidate"
  </FilesMatch>
</IfModule>
```

Naopak tímto zajistíme, že se budou HTML a HTM soubory ukládat po dobu 2 dní s tím, že při každém požadavku dojde k ověření jejich čerstvosti. Pokud se na stránce něco změní, data se opět stáhnou.

Můžeme také využít kratšího zápisu s řízením podle typu souborů, pro obrázky:

```
<IfModule od_expires.c>
  ### aktivace modulu mod_expires
  ExpiresActive On

  ExpiresByType image/gif image/jpg image/jpeg image/png "access
     plus 2 month"
</IfModule>
```

[12]

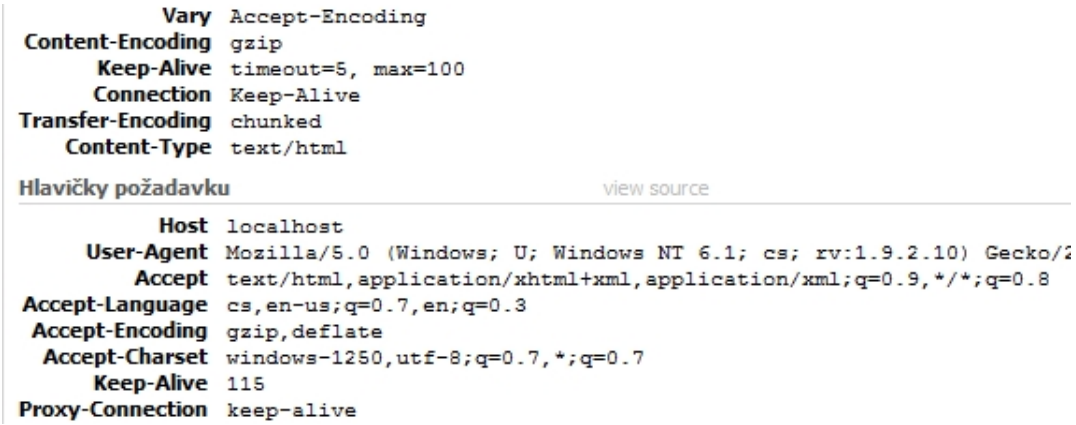
4.3.2.2 Komprimace obsahu

Protokol HTTP 1.1 [41] dovoluje přenášet data v komprimované podobě, díky čemuž je možné snížit objem přenášených dat a zvýšit rychlost načítání webu. Vše opět vychází z nastavení webového serveru, kde je obsah před odesláním komprimován a odeslán do prohlížeče, který jej již zpracuje.

Dnešní webové prohlížeče nemají s touto komunikací žádné problémy. Tento způsob přenosu obsahu je velice vhodný především pro mobilní weby, neboť umožní snížení veli-

[41] https://www.w3.org/Protocols/rfc2616/rfc2616.html

kosti souborů až o desítky KiB. [42]



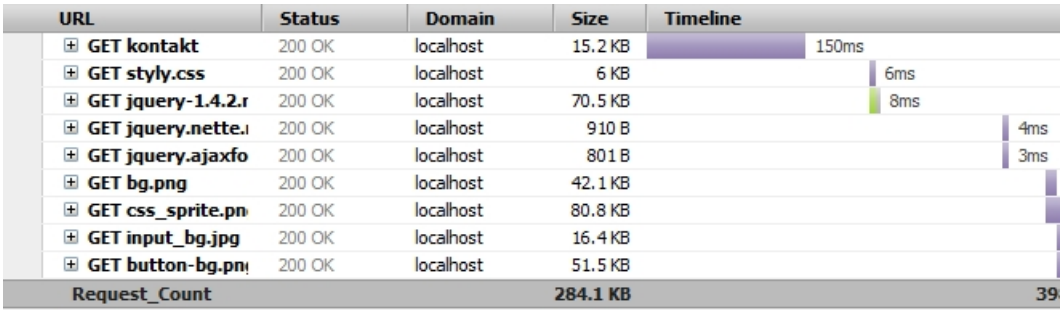
Obrázek 17: Hlavička požadavku od prohlížeče a následná odpověď serveru – data jsou odesílána v komprimované podobě [9]

Na obrázku jsou zachyceny hlavičky požadavku a hlavičky odpovědi. Prohlížeč nejprve serveru řekne, v jaké podobě je schopen komprimovaná data přijmout (Accept-Encoding) a v odpovědi je, že odeslaná data jsou komprimovaná (Content-Encoding). V tomto případě je to pomocí metody *gzip* [43].

Jak ukazují následující dva obrázky, pomocí komprimace obsahu je možné velikost dat znatelně snížit:

[42] V současnosti nastupuje již protokol HTTP 2.0 - http://httpwg.org/specs/rfc7540.html

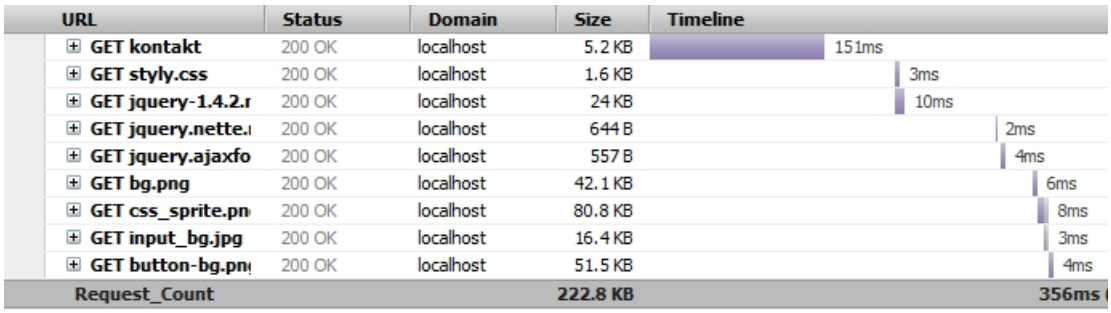
[43] https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer?csw=1



URL	Status	Domain	Size	Timeline
GET kontakt	200 OK	localhost	15.2 KB	150ms
GET styly.css	200 OK	localhost	6 KB	6ms
GET jquery-1.4.2.r	200 OK	localhost	70.5 KB	8ms
GET jquery.nette.r	200 OK	localhost	910 B	4ms
GET jquery.ajaxfo	200 OK	localhost	801 B	3ms
GET bg.png	200 OK	localhost	42.1 KB	
GET css_sprite.pn	200 OK	localhost	80.8 KB	
GET input_bg.jpg	200 OK	localhost	16.4 KB	
GET button-bg.pn	200 OK	localhost	51.5 KB	
Request_Count			284.1 KB	39

Obrázek 18: Stahovaná data ze serveru bez komprese – celkem přeneseno 284,1 KiB [9]

Na obrázku je zachycen stav stažení souborů ze serveru, kde nebyla zapnutá žádná komprese. Celkový objem přenesených dat byl 284,1 KB. Všimněte si především knihovny jquery-1.4.2.min, která má velikost 70,5 KiB.



URL	Status	Domain	Size	Timeline
GET kontakt	200 OK	localhost	5.2 KB	151ms
GET styly.css	200 OK	localhost	1.6 KB	3ms
GET jquery-1.4.2.r	200 OK	localhost	24 KB	10ms
GET jquery.nette.r	200 OK	localhost	644 B	2ms
GET jquery.ajaxfo	200 OK	localhost	557 B	4ms
GET bg.png	200 OK	localhost	42.1 KB	6ms
GET css_sprite.pn	200 OK	localhost	80.8 KB	8ms
GET input_bg.jpg	200 OK	localhost	16.4 KB	3ms
GET button-bg.pn	200 OK	localhost	51.5 KB	4ms
Request_Count			222.8 KB	356ms

Obrázek 19: Stahovaná data ze serveru s kompresí – celkem přeneseno 222,8 KiB [9]

Na dalším obrázku můžete vidět snížení přenášeného objemu dat o 61,3 KiB díky zapnuté kompresi. Jak si můžete povšimnout, velikost knihovny jquery-1.4.2.min je nyní 24 KiB.

Příklad je demonstrován na malé webové stránce [44] s neoptimalizovanými obrázky (mají velkou velikost) a na které komprese nefunguje. Pro obrázky je potřeba další optimalizace.

Podobná situace, jako je u obrázků, se týká i dalšího obsahu, například videa nebo dalších multimediálních souborů. U těch už byla nějaká komprese většinou provedena (i když je jejich velikost stále značná) a další komprese na ně nemá vliv.

Kompresi je tedy vhodné využívat pro všechna „textová“ data, např. *HTML*, *TXT*, *JavaScript*, *CSS* nebo *JSON*.

Nastavení na webovém serveru

[44] http://reg3jez.skauting.cz/

Aby vše správně fungovalo a hlavičky obsahovaly potřebné parametry, je potřeba vše nastavit na webovém serveru. Jako příklad uvedu nastavení pro Apache 2 [45] s využitím modulu mod_deflate [46].

```
<IfModule mod_deflate.c>
  AddOutputFilterByType DEFLATE text/html text/plain text/xml
     text/css application/x-javascript text/javascript
     application/javascript application/json
</IfModule>
```

Touto direktivou nastavíme komprimaci obsahu pro většinu textových dat.

Problém by mohl nastat, pokud by návštěvník webu použil prohlížeč, který nepodporuje HTTP 1.1 (např. starší verze prohlížeče NetScape 4, dnes se však již téměř nepoužívá [47]). Pro tyto případy je nutné nastavit zvlášť celý modul, avšak pro toto nastavení je doporučené pečlivě prostudovat dokumentaci modulu mod_deflate [48].

Jak vypadá komprimovaný obsah?

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="cs">
<head>
  <meta http-equiv="content-Type" content="text/html; charset=UTF-8"/>
  <meta http-equiv="content-Language" content="CS"/>
  <meta name="description" content="Michal Maňák - osobní stránka web designera, web developera, internetc
  <meta name="keywords" content="michal maňák, manakmichal, webdesign, webdevelopment, jquery, blog" /
  <meta name="copyright" content="2010, Michal Maňák"/>
  <meta name="author" content="Michal Maňák"/>
  <meta name="robots" content="index,follow" />

  <link rel="alternate" type="application/rss+xml" title="Blog - články" href="/mm_web/rss/" />

  <link rel="stylesheet" media="all" href="/mm_web/css/styly.css" type="text/css" />
  <script src="/mm_web/js/jquery-1.4.2.min.js" type="text/javascript"></script>
  <script src="/mm_web/js/jquery.nette.min.js" type="text/javascript"></script>
  <script src="/mm_web/js/jquery.ajaxform.min.js" type="text/javascript"></script>

  <title>O mně | manakmichal</title>
```

Obrázek 20: Podoba odeslané HTML stránky bez zapnuté komprese [9]

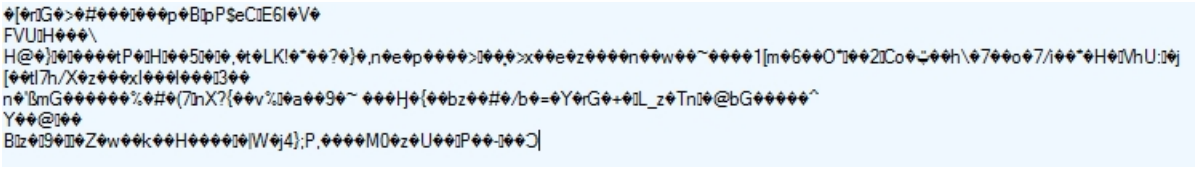
Pokud bychom na serveru neměli zapnutý mod_deflate, stránka by se odeslala v klasické podobě.

[45] http://httpd.apache.org/

[46] http://httpd.apache.org/docs/2.0/mod/mod_deflate.html

[47] https://www.w3counter.com/globalstats.php

[48] http://httpd.apache.org/docs/2.0/mod/mod_deflate.html



Obrázek 21: Podoba odeslané HTML stránky se zapnutou kompresí [9]

Jak ukazuje obrázek, klasický obsah HTML souboru byl změněn na nepřeložitelnou změť znaků, se kterou si však prohlížeč poradí a data převede zpět do správné podoby.

4.3.2.3 Snížení velikosti a počtu externích souborů

CSS sprites

Tato technika je dnes na moderních webových stránkách velice populární. Umožňuje minimalizaci HTTP požadavků v podobě efektivnějšího načítání obrázků na pozadí nebo těch, které nemají významnou informační hodnotu. Nenačítají se totiž všechny obrázky po jednom, ale pouze jedna velká „obrazová mapa“.

Představme si, že máme vytvořenou HTML stránku se seznamem, ve kterém je pro každou položku použita nějaká ilustrující ikona, jak ukazují následující ukázky:

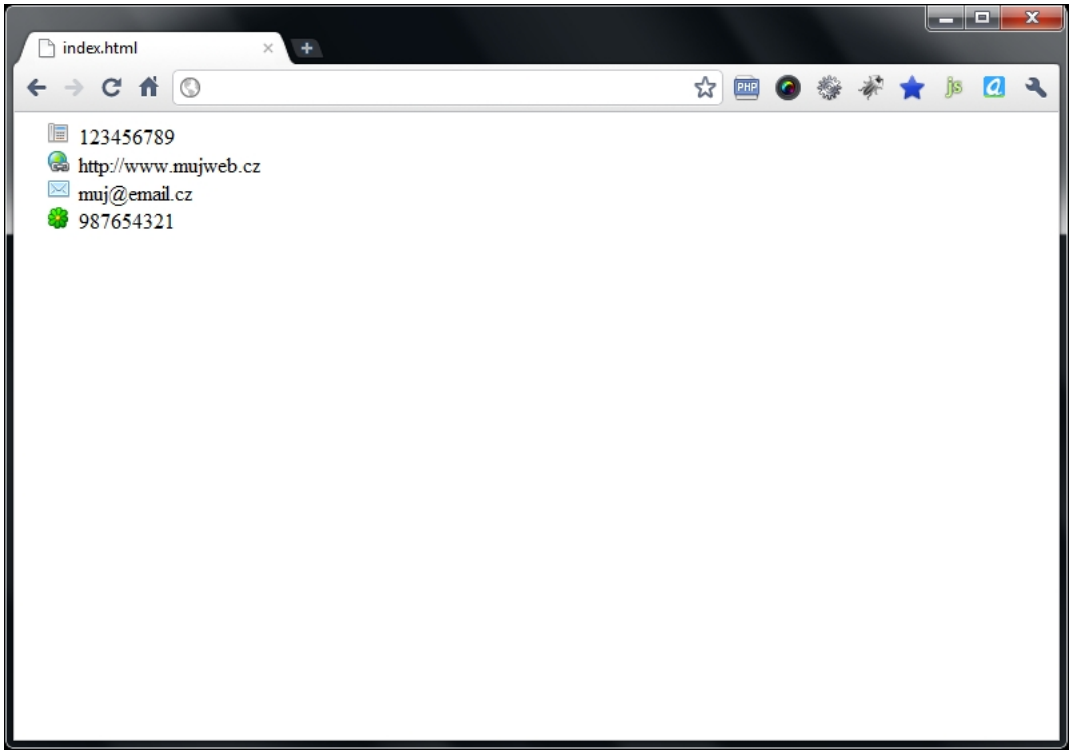
HTML kód:

```
<html>
<head>
  <link rel="stylesheet" href="styly.css" type="text/css"
     media="all" />
</head>
<body>
  <ul>
    <li class="tel">123456789</li>
    <li class="www">http://www.mujweb.cz</li>
    <li class="email">muj@email.cz</li>
    <li class="icq">987654321</li>
  </ul>
</body>
</html>
```

CSS styly:

```
li.tel { list-style-image: url(tel.png); }
li.www { list-style-image: url(www.png); }
li.email { list-style-image: url(email.png); }
li.icq { list-style-image: url(icq.png); }
```

Takto nakódovaná stránka bude v prohlížeči vypadat následovně:



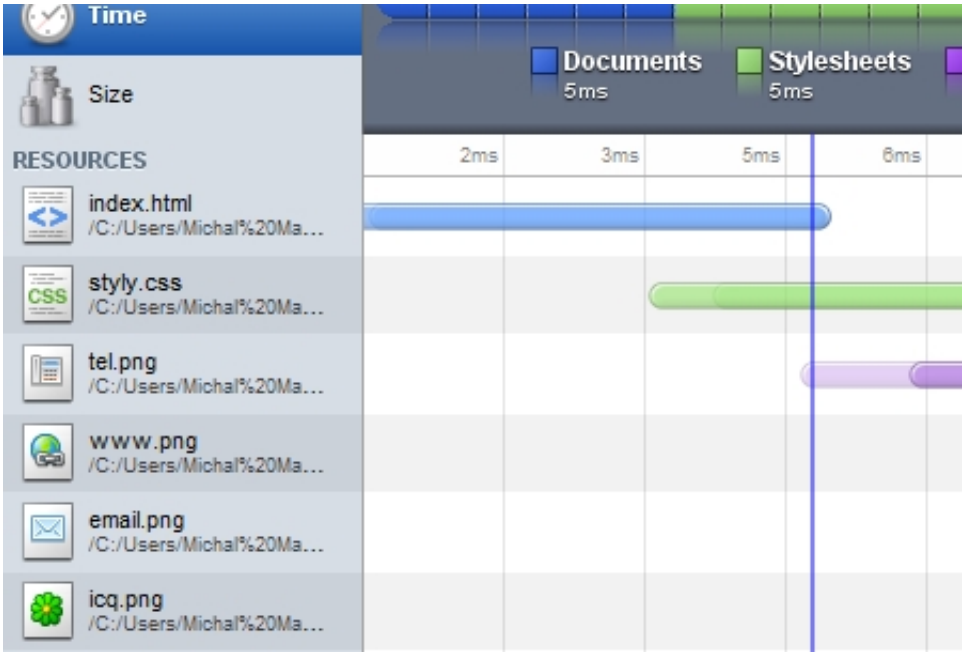
Obrázek 22: Takto nějak může vypadat výsledek výše uvedeného kódu [6]

Výsledek je v prohlížeči jasný a přehledný. Avšak toto řešení vytváří pro každý obrázek jeden HTTP požadavek a tím prodloužení načtení celé stránky.

Jak ukazuje obrázek, celkem bylo vytvořeno 6 požadavků, včetně načtení samotné stránky.

Představte si, že by v seznamu bylo více položek a pro každou by se načítal obrázek zvlášť. To by značně prodlužovalo celkový čas pro načtení celé webové stránky. V tomto případě byl nahrávací čas 9 ms.

Právě díky CSS sprites je možné počet stahovaných obrázků snížit pouze na jeden, který bude obsahovat všechny požadované ikony. Sice bude mít větší velikost, ale celkové načtení stránky bude rychlejší.



Obrázek 23: Ikony se načítaly jednotlivě pro každou položku seznamu [6]

Obrázek 24: Ukázka CSS sprites [9]

V našem případě by bylo potřeba patřičně upravit HTML kód:

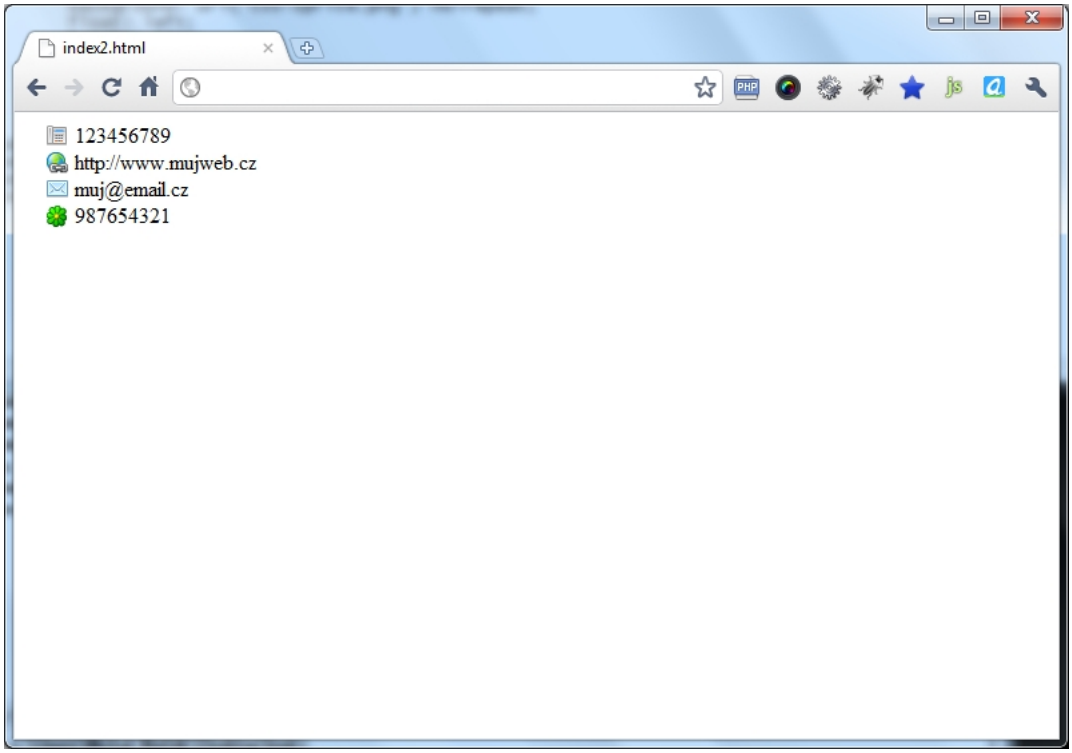
```
<html>
<head>
  <link rel="stylesheet" href="styly2.css" type="text/css"
     media="all" />
</head>
<body>
  <ul>
    <li><span class="tel"></span>123456789</li>
    <li><span class="www"></span>http://www.mujweb.cz</li>
    <li><span class="email"></span>muj@email.cz</li>
    <li><span class="icq"></span>987654321</li>
  </ul>
</body>
```

```
</html>
```

a CSS styly:

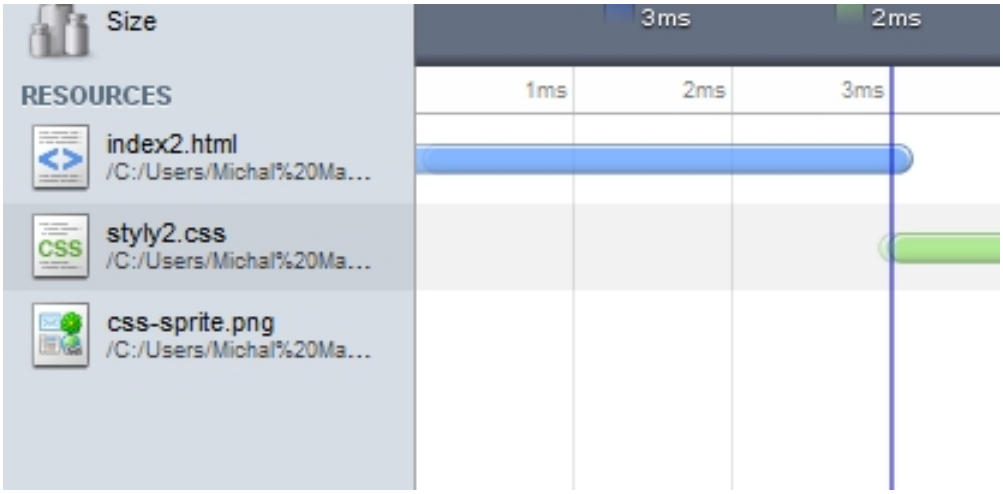
```
ul { list-style: none; padding-left: 16px; }
ul span {
  background: url("css-sprite.png") no-repeat;
  float: left;
  width: 16px;
  height: 16px;
  margin: 2px 5px 0 0;
  overflow:hidden; }
span.tel { background-position: 0 -16px; }
span.www { background-position: -16px -16px; }
span.email { background-position: 0 0; }
span.icq { background-position: -16px 0; }
```

Takto upravená stránka vypadá v prohlížeči stejně jako v předchozím případě:



Obrázek 25: Stránka s využitím CSS sprites vypadá stejně jako bez použití této techniky [9]

Jak ze samotných úryvků kódu vyplývá, celkový počet HTTP požadavků se sníží právě díky sjednocení ikon do jednoho souboru:



Obrázek 26: Celkový počet HTTP požadavků se snížil díky sjednocení ikon do jednoho souboru [9]

Díky CSS sprites došlo ke snížení jak celkového počtu HTTP požadavků, ale také ke snížení celkového času pro nahrání stránky na 7 ms (v předchozím případě 9 ms.).

Tato technika je velice užitečná a dochází k opravdu znatelnému zrychlení načítání webových stránek. Dnes je standardem a využívají ji všechny moderní webové stránky a aplikace, konkrétně například Twitter [49], Google [50] nebo Facebook [51].

Obrázek 27: Obrazová mapa, kterou pro CSS sprites používal Twitter [9]

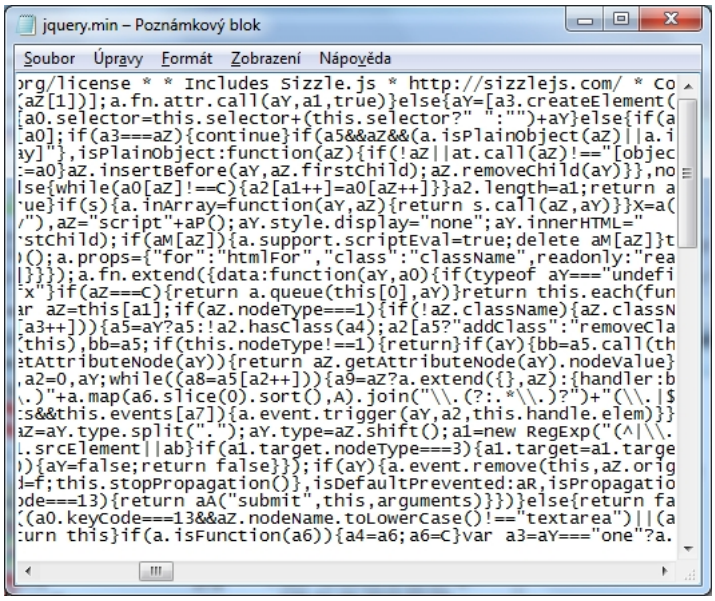
[49] https://twitter.com/
[50] https://google.com/
[51] https://facebook.com/

4.3.2.4 Minimalizace CSS a Javascriptu

Webové stránky také neodmyslitelně tvoří externí soubory, například CSS styly nebo soubory s JavaScriptem. Soubory se nahrávají a zpracovávají vždy před samotným vykreslením stránky. Proto je důležité dbát na jejich co nejmenší velikost.

Existuje několik způsobů, jak lze zmenšit tyto externí soubory i o několik desítek procent a zrychlit tak načtení samotné stránky.

Nejběžnějším a nejpoužívanějším způsobem je tzv. **minifikace**. Její princip spočívá v odstranění mezer a tabulátorů, tzv. *bílých znaků*. Tímto způsobem lze snížit velikost CSS stylů a JavaScriptu. Například lze minifikovat javascriptový soubor o velikosti 36 KiB na velikost 15 KiB – což je ve výsledku snížení o více jak 41 %.

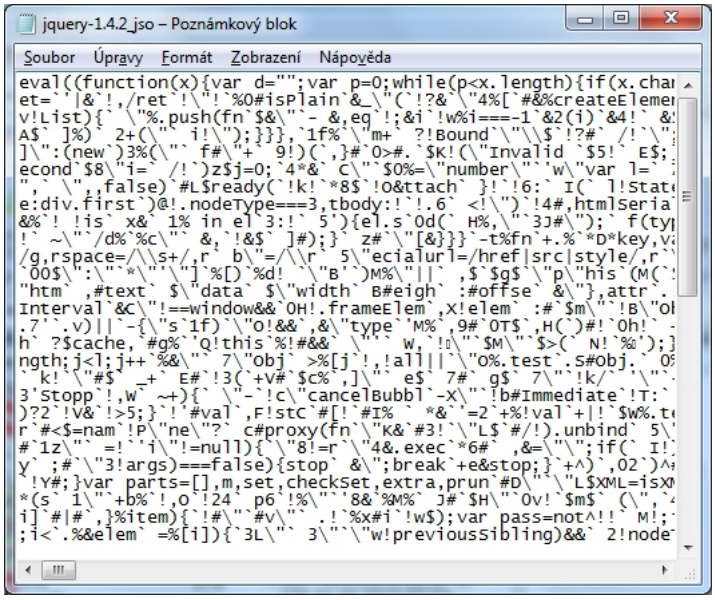


Obrázek 28: Minifikovaná produkční verze souboru jQuery 1.4.2 [9]

Dalším způsobem je tzv. **obfuskace**. Její princip spočívá v převedení souboru na jednoduchý zápis se zapojením co nejjednodušších identifikátorů. Tímto způsobem lze snížit velikost javascriptových kódů. Například produkční verze jQuery o velikosti 163,8 kB byla snížena na 59,4 kB a v případě minifikace byla velikost výsledného souboru 79,7 KiB. [52]

Metoda obfuskace má však nevýhodu v tom, že může v některých případech generovat chyby a v „přeformátovaném“ souboru se chyby opravdu špatně opravují. Proto je potřeba

[52] Vyzkoušeno pouze demonstrativně. jQuery na svém webu poskytuje ke stažení i minifikované verze.



Obrázek 29: Obfuskovaná produkční verze souboru jQuery 1.4.2 [9]

dávat pozor na takto „komprimované“ soubory.

Nástroje

Ke zmíněným způsobům jsou určené vhodné nástroje, které snížení velikosti externích souborů usnadní.

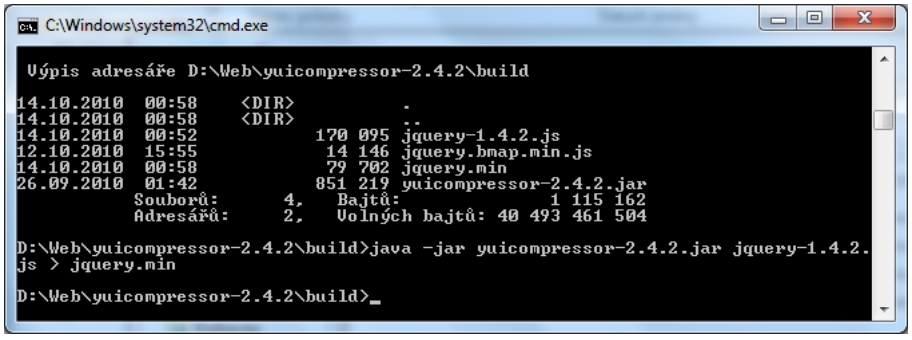
Pro CSS:

- YUI Compressor,
- CSS Compressor.

Pro JavaScript:

- YUI Compressor,
- Google Closure Compiler,
- JavaScript Obfuscator.

Nástrojů je samozřejmě více, tyto nástroje zmiňuji pouze proto, že jsou použity v kontextu této práce.



Obrázek 30: YUI Compressor je možné používat v příkazovém řádku Windows [9]

4.3.3 Optimalizace obrázků

4.3.3.1 Formáty obrázků pro web

Jedním z nejvyužívanějších typů souborů na webu jsou obrázky. Díky nim je přenášený objem dat poměrně vysoký, avšak použitím vhodného formátu je možné objem dat držet poměrně nízko.

Nejpoužívanější formáty obrázků na internetu jsou:

- GIF
- PNG
- JPG (JPEG)

Díky oblíbenosti zařízení typu mobil/tablet, které mají obrazovky s vysokou hustotou pixelů, se začaly rozšiřovat i vektorové formáty [53].

Výše uvedené formáty obrázků mají specifické vlastnosti a jsou určené pro různé použití. Jejich správným použitím lze zajistit nejen kvalitu zobrazení, ale také snížit objem přenášených dat.

GIF – Graphics Interchange Format

GIF (*Graphics Interchange Format*) je grafický formát určený pro rastrovou grafiku. Nabízí maximálně 8 bitovou barevnou hloubku (256 barev) a bezztrátovou kompresi. Umožňuje omezené možnosti průhlednosti. Výhodou GIF je především možnost jednoduchých animací.

[53] https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Adding_vector_graphics_to_the_Web

PNG – Portable Network Graphics

PNG (*Portable Network Graphics*) je grafický formát nabízející bezztrátovou kompresi. Nabízí možnost 24 bitové barevné hloubky a 8 bitovou průhlednost. Je zde také formát nabízející 256 barev včetně transparence, tzv. adaptivní PNG8. Zatím není stanoven formát pro jednoduché animace.

Tento formát je nástupcem zastaralého typu GIF.

JPG (JPEG)

JPG, nebo spíše přesněji JPEG, je grafický formát určený pro prezentaci fotografií. Funguje na základě ztrátové komprese, takže je možné dosáhnout foto-realistické kvality i s malou velikostí souboru.

Obsahuje také další informace o obrázku, tzv. EXIF metadata [54], například:

- Rychlost závěrka
- Typ fotoaparátu
- GPS pozice (pokud je k dispozici)
- Světelnost
- Informace o miniatuře (zmenšenině / náhledu)

Formát JPEG nepodporuje možnost průhlednosti.

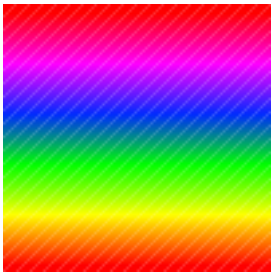
Srovnání formátů

Pro ilustraci kvality a efektivity jednotlivých formátů obrázku jsem připravil tabulku, která porovnává jejich velikost.

Protože na první pohled se obrázky zdají naprosto stejné, uvádím zde pouze ilustraci. Samozřejmě až při bližším zkoumání jsou znát patrné rozdíly v kvalitě. Na obrázcích je barevné spektrum obsahující celkem 188 barev. Obrázek označený červeně má největší velikost, obrázek označený zeleně má nejnižší velikost.

Největší rozdíly jsou ve velikosti. Nejvyšší velikost má obrázek typu JPEG bez komprese (20464 B) a nejnižší PNG24 (1292 B). Formát GIF nabízí v porovnání s PNG více jak trojnásobnou velikost souborů.

[54] https://en.wikipedia.org/wiki/Exif



Formát obrázku	Velikost
GIF – paleta obsahuje přesně 188 barev	4357 B
GIF (adaptivní) – paleta nabízí 256 barev (188 skutečně využitých)	4357 B
GIF – paleta obsahuje 128 barev	3714 B
JPEG – 25% komprese (75% poměr zachování kvality)	2801 B
JPEG – bez komprese JPEG	20464 B
PNG8 – paleta nabízí 256 barev (188 skutečně využitých)	1319 B
PNG24 – paleta nabízí 16,7 milionů barev	1292 B
PNG32 – paleta nabízí stejný počet barev jako PNG24 + 8 bitů pro průhlednost	1454 B

Tabulka 1: Tabulka srovnání formátů

Průhlednost

Dnes je na webu zcela běžné používat obrázky v podobě ikon nebo ilustrujících znaků. Ty jsou specifické tím, že se dají univerzálně používat na jakémkoliv pozadí – jsou na pozadí průhledné.

Jediné dva formáty, které nabízejí průhlednost obrázku, jsou **GIF** a **PNG**. Avšak oba formáty mají různé vlastnosti a průhlednost u nich vypadá jinak.

Protože GIF nabízí pouze 256 barev (8 bitů), nemůže nabídnout tak kvalitní vlastnosti průhlednosti jako PNG, který umožňuje průhlednost o hloubce 8 bitů.

Starší prohlížeče, například Internet Explorer 6, průhlednost u PNG nepodporují a často se to řeší pomocí GIF. Jsou zde však možnosti, jak i pro tento prohlížeč vynutit průhlednost u PNG, například:

- jQuery.pngFix.js
- IE PNG FIX

- Unit PNG Fix

Optimalizace obrázků

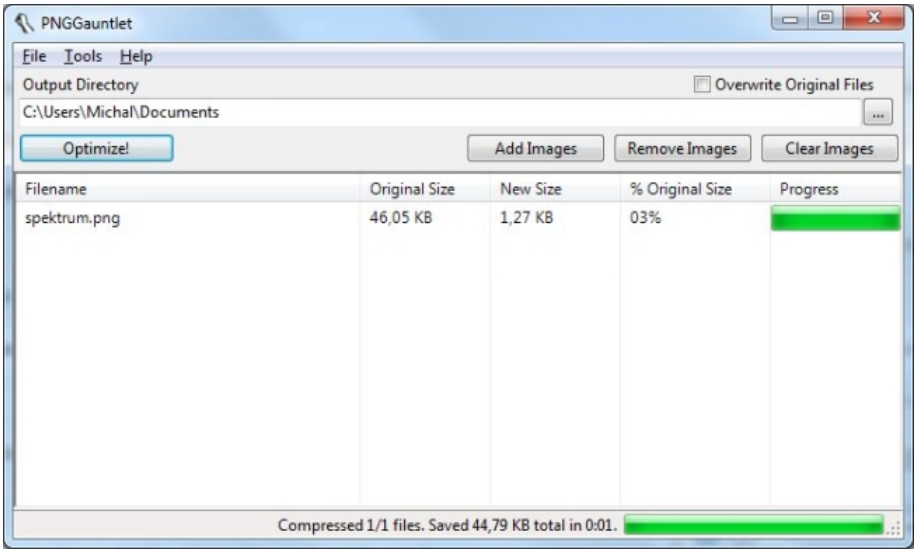
Zvolením vhodného grafického formátu ještě není vyhráno. Pokud jsou obrázky vytvořené v grafickém editoru a nejsou uloženy efektivním způsobem, jejich velikost stále bude zbytečně velká – především díky vloženým komentářům samotného editoru.

Proto je potřeba obrázky ještě dostatečně optimalizovat některým programem (většinou to samy grafické editory nenabízejí), například:

- PNGcrush – PNG
- PNGGauntlet – PNG
- ImageMagick – PNG, JPEG
- IrfanView s pluginy RIOT a PGNOUT – PNG, JPEG
- Linuxová knihovna jpegtran – JPEG
- Webová služba JPEGmini – JPEG

Optimalizace PNG

Nejefektivnějším způsobem, jak optimalizovat PNG obrázky, je využít program *PNGGauntlet*. Ten nabízí nejvyšší možnou míru „stlačení“ velikosti, kterou jsem u různých programů zaznamenal.

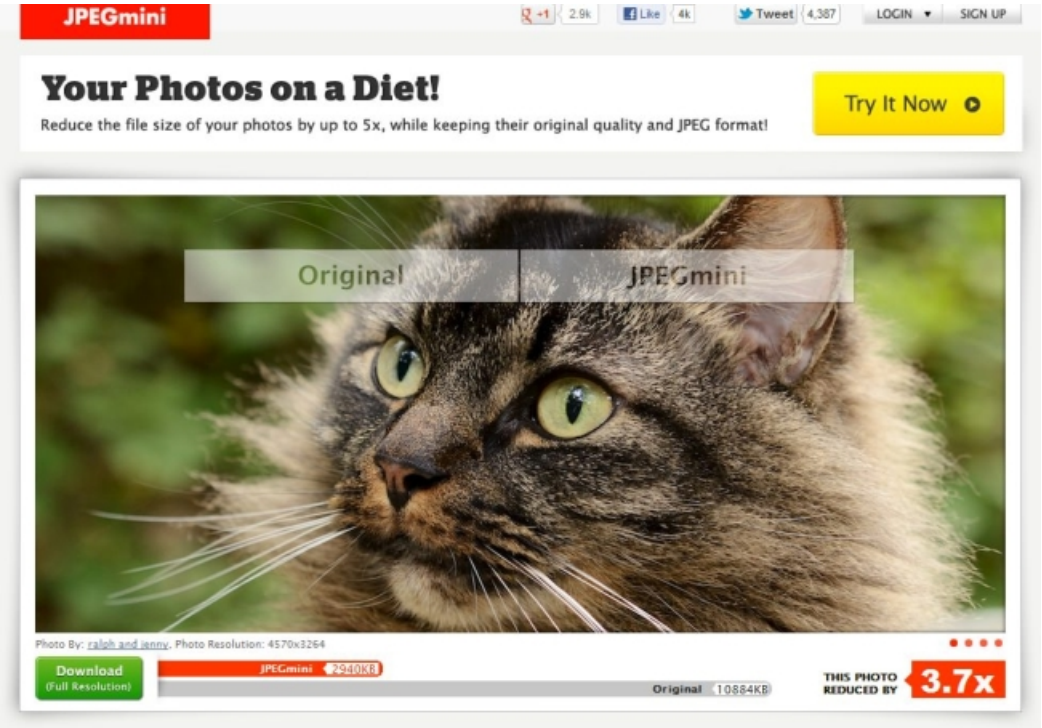


Obrázek 31: JPEGmini nabízí možnost srovnání původní a upravené fotografie [9]

Optimalizace JPEG

Obrázky ve formátu JPEG je vhodné optimalizovat volbou určité míry komprese. Nejlepší výsledky jsem dosáhl pomocí programu *Irfanview* a kompresí v rozmezí 75 – 80 %, kdy téměř nebylo možné rozeznat snížení kvality.

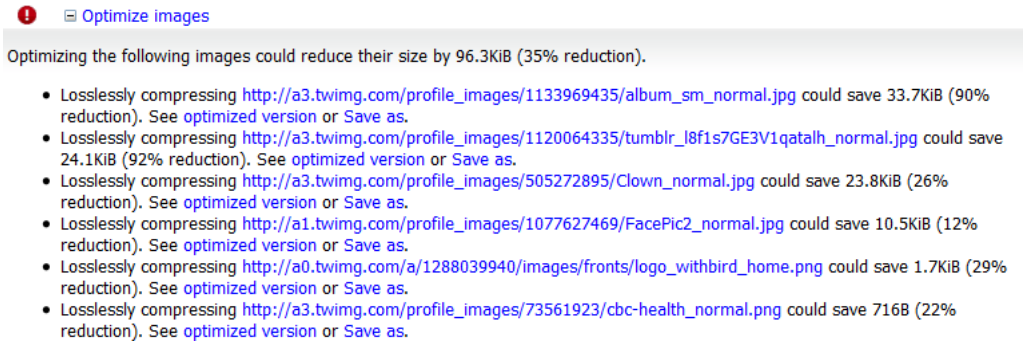
Velmi efektivní je také webová služba *JPEGmini*, která nabízí snížení velikosti fotografií téměř bez ztráty kvality. Při bližním zkoumání jsem však dosáhl stejných a lepších výsledků, než jaké služba nabízí, právě volbou komprese 75 – 80 %.



Obrázek 32: JPEGmini nabízí možnost srovnání původní a upravené fotografie [9]

Další možnosti

Díky rozšíření Google PageSpeed [55] lze obrázky získat v optimalizované podobě. Stačí si rozšíření nainstalovat do prohlížeče Google Chrome nebo Firefox a provést analýzu webu. Tím lze získat přehled nejen o stavu obrázků, ale o dalších problematických částech.



Obrázek 33: Google PageSpeed umožňuje stažemé optimalizovaných obrázků na vašem webu [9]

[55] https://developers.google.com/speed/pagespeed/

Optimalizace CSS sprites

Pro CSS spirtes potřebujete vytvořit obrázek, který bude mít velké rozměry a nebude patřit mezi ty s nejmenší velikostí. Proto je potřeba zvolit:

- Správný formát,
- Orientaci obrázku.

Formát obrázku	Velikost
PNG32 – horizontální orientace	4289 B
PNG8 – horizontální orientace	2306 B
PNG32 – vertikální orientace	4214 B
PNG8 – vertikální orientace	2306 B

Tabulka 2: Tabulka srovnání formátů CSS sprites [9]

Jak ukazuje srovnávací tabulka, použitím různých formátů a orientací můžete získat různou velikost obrázků.

V případě použití jednoduchých obrázků ve formě ikon je vhodnější použít formát PNG8 i přes menší počet barev – rozdíl není rozeznatelný a výsledná velikost je téměř 2x nižší.

V případě volby orientace vychází lépe vertikální orientace, tedy do sloupce. Byla vyzkoušena tvorba mnoha optimalizovaných CSS sprites a jak ukazuje tabulka u formátu PNG32, i při použití stejných obrázků získáváme výslednou nižší velikost souboru.

4.3.3.2 Komprese HTML

Minimalizace HTML

Hlavním cílem minimalizace HTML je snížit objem přenášených dat. To je zajištěno díky odmazání nepotřebných znaků v podobě mezer a tabulátorů. Může vyvstat otázka, zda je minimalizace HTML skutečně potřeba.

Postup jsem však testoval v několika variantách na různých projektech a došel jsem k několika závěrům:

- Ušetří se přenášená data.
- Rychlejší vykreslení v prohlížeči.

HTML je jednou z nejpodceňovanější součástí každého webu. Často je k vidění „plýtvání" značkami, což vytváří větší nároky na vykreslení (i v době, kdy jsou vykreslovací jádra prohlížečů velmi rychlá).

Úspora přenesených dat

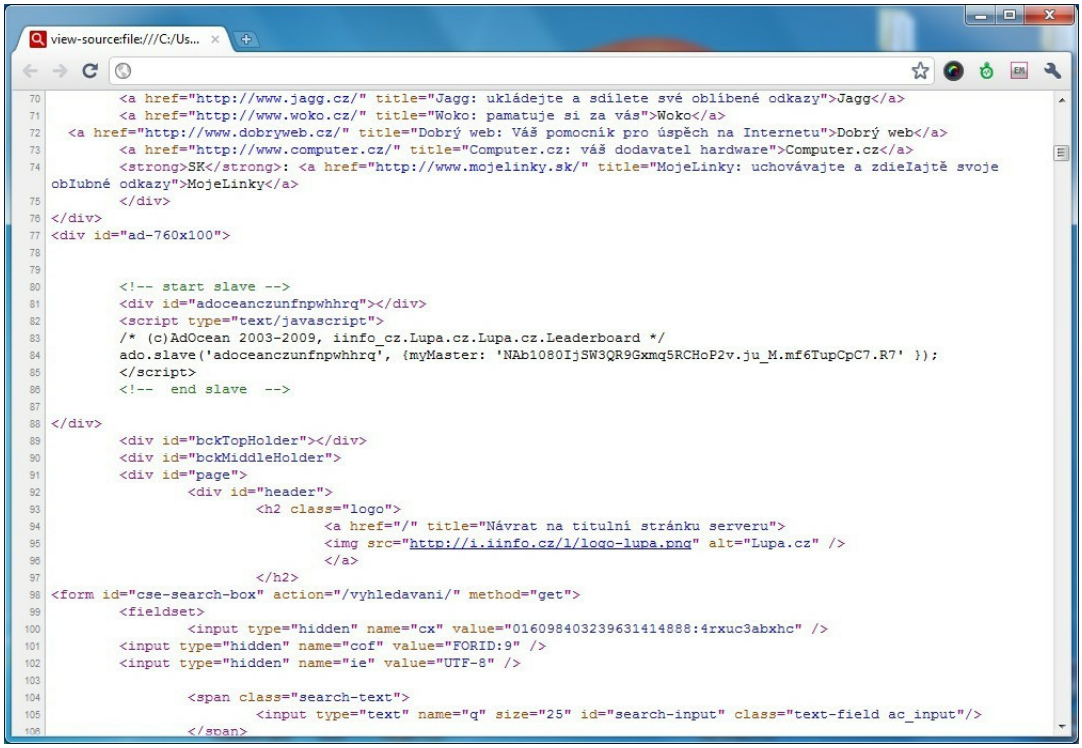
Příkladem je webová aplikace, která je předmětem této práce a na které byly provedeny výše uvedené úpravy.

HTML kód v neupravené podobě má celkovou velikost 53775 B a obsahuje velké množství „bílých znaků". Pokud by se všechny tyto nechtěné mezery odstranily, stránka by se zmenšila na na 50063 B (tedy cca 48,8 KiB). Dosáhlo by se úspory 3712 B.

V případě velké návštěvnosti, například 1000 lidí denně, by úspory na přenášených datech činily 3,7 MB.

Při využití komprese by rozdíl v přenášených datech nebyl tak markantní, avšak je zde další důležitý ukazatel – rychlost vykreslení.

Rychlejší vykreslení



Obrázek 34: Úvodní stránka s neupraveným HTML [9]

Důležitějším faktorem, proč minimalizovat HTML kód, je zrychlení načítání stránky v samotném prohlížeči, což lze vidět například použitím prohlížeče Google Chrome se zapnutým módem Speed Tracer a Developer Tools.

Jak můžete vidět na obrázku, vykreslení stránky s neupraveným HTML trvalo prohlížeči déle než 2,6 sekundy. Zpracování HTML kódu pak trvalo přibližně 237 ms (bez započítání samotného vykreslení s úpravou přes JS nebo CSS).

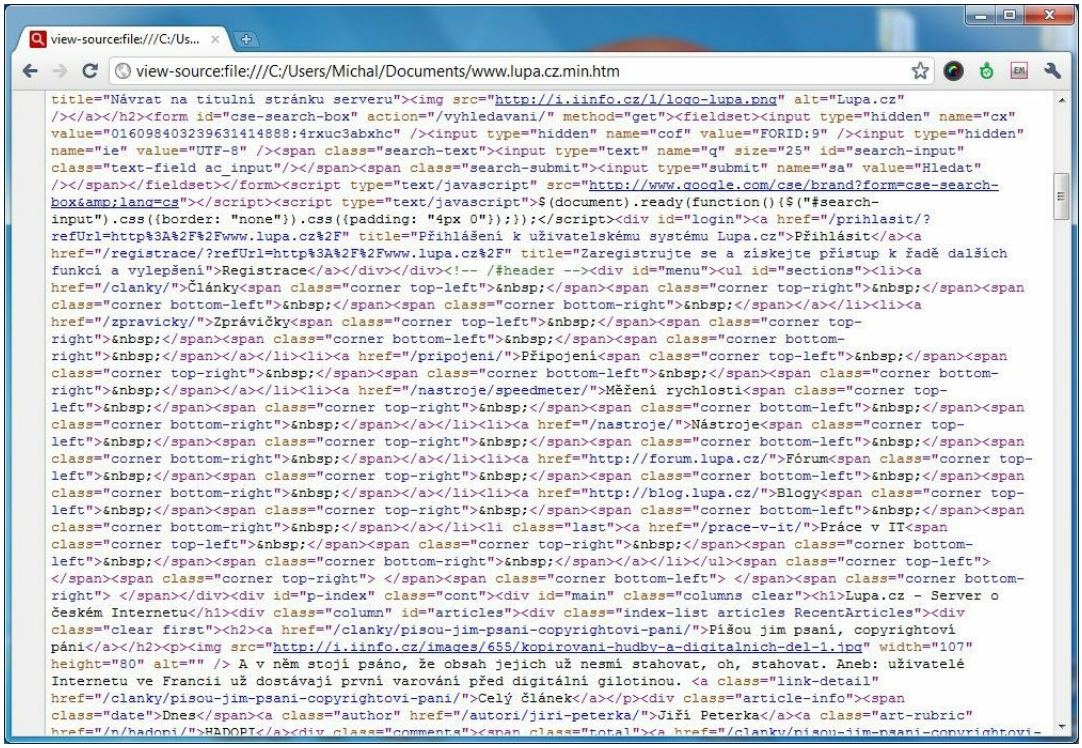
Při minimalizaci HTML by došlo k viditelnému zrychlení zpracování HTML. Prohlížeč projde jednotlivé značky postupně bez nutnosti procházení a přeskakování mezer:

Jak můžete vidět na obrázku, vykreslení stránky s minimalizovaným HTML trvalo prohlížeči méně jak 2 sekundy. Zpracování HTML kódu pak trvalo přibližně 198 ms (bez započítání samotného vykreslení s úpravou přes JS nebo CSS).

Celkový rozdíl ve vykreslení je pak více jak 0,5 sekundy, což už je rozdíl 30 %.

Externí soubory

Nesprávným odkazováním na externí soubory se může doba potřebná na vykreslení



Obrázek 35: Úvodní stránka s minimalizovaným HTML [9]

stránky protáhnout i v řádech sekund. Důvodem může být nejen samotné odkazování, ale také množství souborů, na které se odkazuje.

Nejvíce problematické jsou JavaScripty. Ty jsou při svém stažení prohlížečem ihned analyzovány (parsovány) a provedeny. JavaScript totiž může u aplikace změnit:

- Vzhled,
- Chování.

Po stažení je veškeré další stahování pozastaveno, dokud není JavaScript zpracován. Tím se pozastavuje vykreslování a formátování stránky pomocí CSS, stažení dalších JavaScriptových souborů, atp.

Je tedy potřeba, aby byl JavaScript odkazován jako poslední z externích souborů.

Odkazování v hlavičce

Pokud budou JavaScriptové soubory odkazovány jako první, nebo střídavě s CSS styly, načtení samotné stránky se prodlouží.

Na ukázku jsem připravil jednoduchý web, kde jsou odkazovány soubory v tomto pořadí:

- CSS

- JavaScript
- JavaScript
- CSS
- JavaScript

Využil jsem standardní knihovny jQuery, jQuery UI, Modernizr a předpřipravené CSS.

Na obrázku můžete vidět, že po stažení prvního JavaScriptu se veškeré další stahování pozastavilo až do jeho zpracování. Ten však může být paralelně stažen s CSS styly. Celkově se web načítal 3,75 sekund.

Jak se změní čas potřebný pro nahrání webu, pokud bychom změnili pořadí na:

- CSS
- CSS
- JavaScript
- JavaScript
- JavaScript

Při správném odkazování je možné dosáhnout výrazného zrychlení načítání webu. Jak můžete vidět na obrázku, nahrání webu trvalo 2,68 sekund.

Rozdíl je nejen v samotném načtení webu – 1,07 sekund, ale i ve zpracování souborů – přibližně 240 ms. [20]

Další způsoby odkazování

Několikrát jsem se setkal s řešením, kdy byly odkazy na externí soubory přiřazovány ne zrovna vhodným způsobem, například:

- JavaScript pomocí zápisu `document.write('<script><\/script>');`
- CSS styly pomocí `@import url('');` uvnitř souboru s CSS styly.

V těchto případech prohlížeč neví, co je obsahem odkazovaného souboru a veškeré stahování a zpracovávání zastaví až do doby, než bude odkazovaný soubor načten a zpracován.

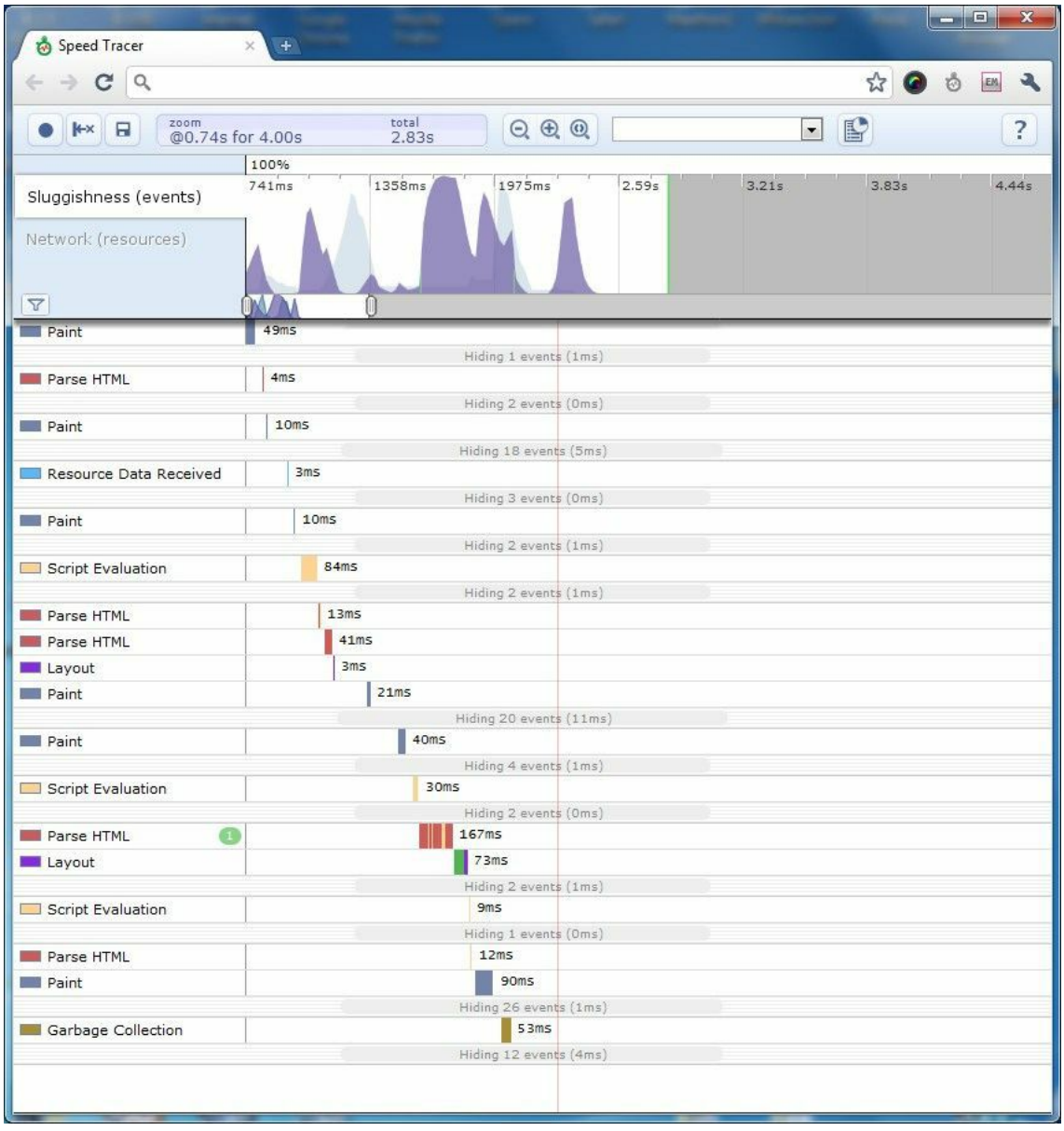
Pokud bychom přikládali externí soubory zmíněným JavaScriptovým zápisem, nejprve by se musel zpracovat první JavaScriptový blok – vypsání `<script>` tagu - a následně by se zpracovával další JavaScript.

Na obrázku můžeme vidět, že i když jsou soubory správně seřazené, zpracování souborů trvalo 2,14 sekund. V porovnání s předchozím příkladem (1,68 sekund) se tak čas potřebný pro zpracování prodloužil téměř o 0,5 sekundy.

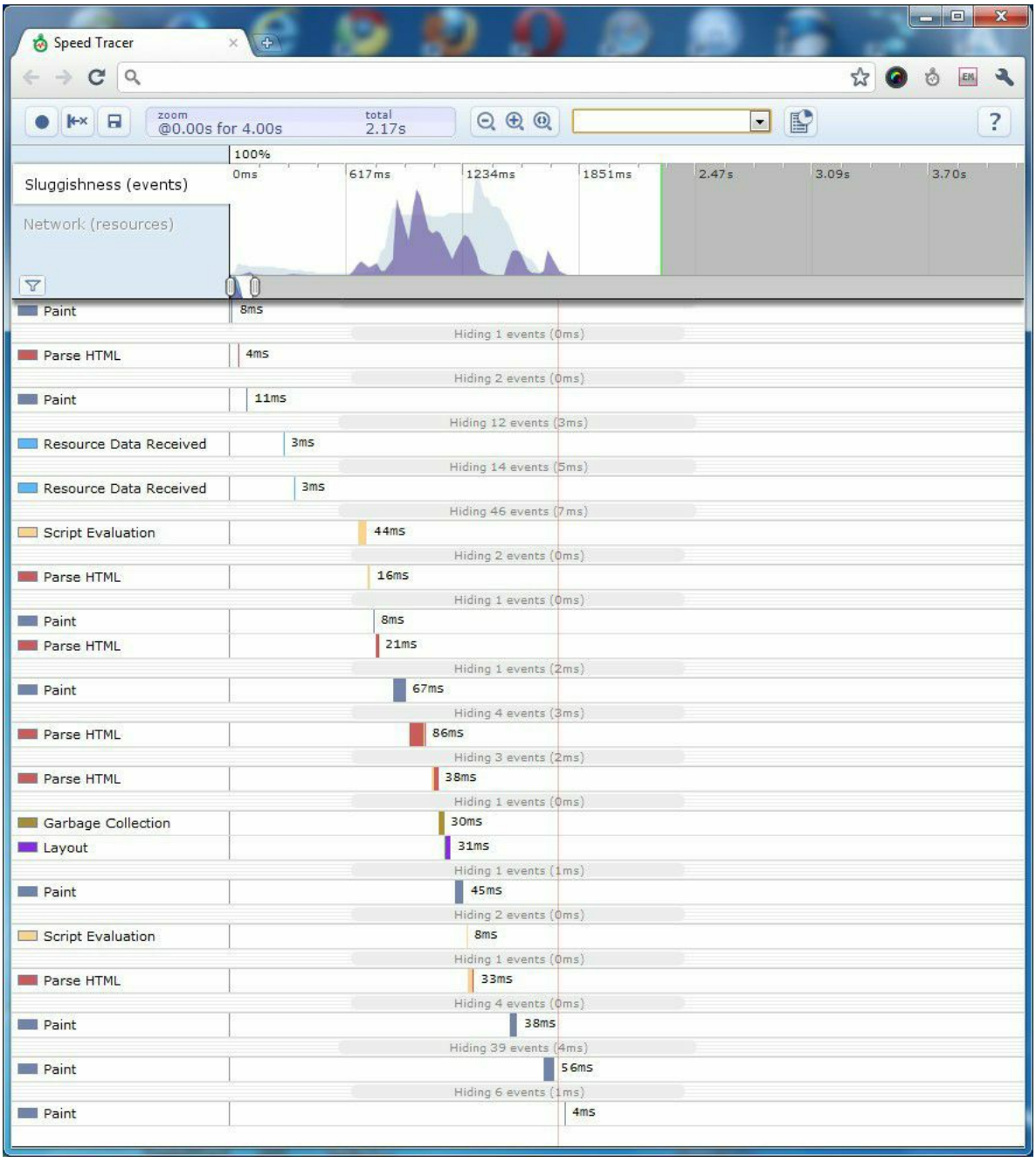
V případě použití `@import url('');` v těle CSS stylů bude potřeba tyto odkazované soubory stáhnout a zpracovat. Bude vytvořen další HTTP požadavek a odkazovaný soubor bude samostatně zpracováván – vše ostatní bude pozastaveno.

Pokud uvedeme odkazy v hlavičce stránky ve správném pořadí, vnořený odkaz ve formě `@import` způsobí, že se po vykonání předchozích požadavků pozastaví vše ostatní až do doby, než bude odkazovaný soubor zpracován.

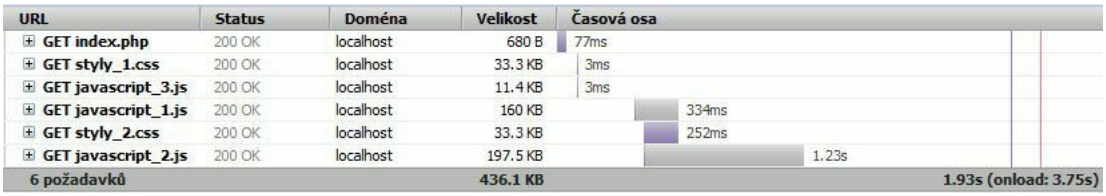
To vede opět k prodloužení načítání. V tomto případě byl čas potřebný na zpracování externích souborů 1,9 sekundy (nahrání celé stránky trvalo 2,93 sekund). V porovnání se správným řešením (1,69 sekund pro zpracování a 2,68 sekund) tak došlo opět ke zpomalení nahrávání přibližně o 300 ms.



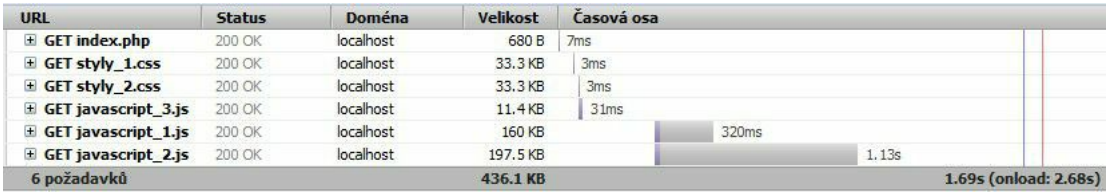
Obrázek 36: Vykreslení úvodní stránky s neupraveným HTML trvalo více jak 2,6 sekundy [9]



Obrázek 37: Vykreslení úvodní stránky s minimalizovaným HTML trvalo více jak méně jak 2 sekundy [9]

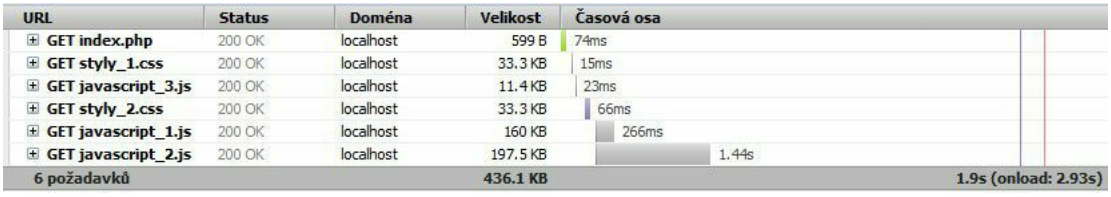


Obrázek 38: Načtení webu se střídavým odkazováním trvalo 1,93 sekund (celkově pak 3,75 sekund) [9]



Obrázek 39: Načtení webu se správným odkazováním trvalo 1,69 sekund (celkově pak 2,68 sekund) [9]

Obrázek 40: Správné pořadí odkazovaných souborů, JS soubory jsou však odkazován pomocí JavaScriptu v těle stránky [9]



Obrázek 41: Soubory jsou odkazovány ve správném pořadí, avšak styly_1.css obsahuje odkaz na další soubor se styly [9]

5 Zhodnocení výsledků

Aplikace, respektive uživatelské rozhraní a jeho vývoj, což je stěžejním předmětem této práce, je poměrně velmi komplexní a rozsáhlé. Takto rozsáhlé rozhraní nelze jednoduše realizovat bez předchozích příprav a konzultací. V přehledu dané problematiky se tato práce tedy zabývá hlavně tím, jak k danému vývoji přistoupit pomocí metodiky pro vývoj softwaru tak, aby byl daný software realizován v daném rozsahu, času a kvalitě. Na systém je tak nahlíženo z pohledu několika různých metodik a je pak také definováno, kdo a jakým způsobem se na návrhu webu/aplikace podílí.

Vlastní práce se již snaží dle konkrétní metodiky Scrum realizovat frontend aplikace za pomoci nástrojů, které jsou k tomu k dispozici. Potřeby a procesy uživatele jsou tak zmapovány, dekomponovány a převedeny do map a vlastností uživatelského rozhraní. Celé uživatelské rozhraní je takto zanalyzováno, aby poté mohlo být realizováno jako drátěný model a následně prototypováno pomocí frontendových frameworků. Zadavatel i vývojář mají tak možnost se k návrhu neustále vracet a v průběhu ihned řešit vyvstalé problémy či komplikace. Díky frameworkům je tak možné mít v poměrně krátkém čase fungující uživatelské rozhraní.

Přístup k vývoji uživatelského rozhraní pomocí iterací a prototypování umožnil odhalovat chyby v návrhu již od počátku. Právě relativně úzká spolupráce se zadavatelem a uživatelem a úpravy aplikace po malých a rychlých iteracích a použití funkčního prototypu umožnily aplikaci prakticky ihned používat a testovat. Došlo tak ke včasnému odhalení chyb v návrhu již v počátku. Jiné nepřesnosti pak byly odhalovány a odstraňovány prakticky ihned v průběhu vývoje.

Pro zjednodušení a urychlení návrhu a vývoje bylo uvažováno i použití frontendového frameworku. Vzhledem k rozšířenosti HTML a CSS bylo k výběru frameworků hned několik. V potaz však byl brán jenom pouze ten s jednoduchou a jasnou strukturou, dobrou dokumentací a aktivní podporou. To vše se sešlo u frameworku Bootstrap, u kterého byla výhodou hlavně jednoduchost a rozsáhlá podpora komunity.

Framework samotný pak v případě vývoje opravdu urychluje a zjednodušuje postup prací a realizaci jednotlivých komponent a stanovuje jasnou strukturu projektu. Díky dobré učící křivce se průběh vývoje systému postupně zrychluje.

Toto pozitivum frameworku je však také zároveň jeho negativem. V případě implementace řešení u velmi komplexních aplikací principy a komponenty frameworku již přestávají stačit. Webdesigner pak musí framework obcházet a ještě častěji hledat vlastní řešení, neboť framework řeší pouze nejčastější případy užití.

Vlastní řešení se ze své podstatné části také dále zabývalo optimalizací rychlosti aplikace. Jak již byl na začátku práce zmíněno předpokládá se, že real-time chování aplikace v tomto případě znamená dostatečně rychlou a neblokující odezvu na jakýkoli uživatelský požadavek. Tedy že uživatel nemá pocit, že by musel čekat nebo že by web reagoval pomalu. Samotná optimalizace sestávala s několika konkrétních kroků, které se podařilo všechny realizovat a díky kterým aplikace doznala citelného zrychlení v řádu sekund, což opravdu zvýšilo uživatelský komfort a nejdůležitějšího cíle této práce tak bylo dosaženo.

Webová aplikace řešící uživatelské rozhraní pro registraci na závod Napříč Prahou – přes 3 jezy tak v zásadě splňuje vytyčené cíle. Nejsou však realizovány všechny uživatelské procesy s požadovanými vlastnostmi a funkcionalitami, avšak vše je definováno pro další vývoj i rozšiřování do budoucna.

6 Závěr

Informační systém a jeho frontendová část, jež byla předmětem této práce, byla představena již v mé bakalářské práci. V rámci této práce došlo k jejímu dalšímu vývoji a rozšiřování tak, jak bylo předesláno. Oproti předchozímu vývoji se stala hlavním tématem frontendová část, tedy to, co vidí uživatel a přes co s celým systémem komunikuje.

Cíle této práce byly stanoveny obecně a umožnili tak definovat aplikaci v širším kontextu. Opět je tak dokázáno, že vývoj takového systému je týmová práce pro více lidí, neboť je rozsáhlejší, než bylo očekáváno. I přesto realizované části frontendového řešení dokazují, že se vyplatí navrhovat s rozumem a navržené řešení otestovat s reálnými uživateli. K dobrému uživatelskému cítění taktéž přispívá znatelně optimalizovaná rychlost načítání, kterou i běžný uživatel pocítí.

Nebýt podpůrných prostředků, jakými jsou frontendové frameworky, nástroje pro projektové řízení či vývojové metodiky softwaru, zajisté by se nepodařilo realizovat práci v takovém rozsahu a kvalitě.

I přes přípravu a podpůrné prostředky nelze takto komplexní systém vyvinout ihned. Práce staví na již připraveném základu a dále ho rozšiřuje a otevírá další cestu k možným realizacím a rozšiřování stávajícího systému, který závodníkům zpříjemňuje registraci a organizátorům usnadňuje hodně práce.

7 Seznam použitých zdrojů

Literatura

[1] Adam, K.: Design webů v prohlížeči. 2015-09-21, [online].
URL <http://www.vzhurudolu.cz/blog/38-design-v-prohlizeci>

[2] Alistair, C.: *Use Cases - Jak efektivně modelovat aplikace*. Brno: Computer Press, a.s., 2005, ISBN 80-251-0721-3.

[3] Cederholm: *Webdesign s webovými standardy*. Brno: Zoner Press, a.s., 2004, ISBN 80-86815-15-3.

[4] dolů, V.: Rozhovor: S Adamem Kudrnou o Bootstrapu. 2016-03-16, [online].
URL <http://www.vzhurudolu.cz/blog/55-adam-kudrna-rozhovor>

[5] Řezáč Jan: *Web ostrý jako břitva*. Praha: Baroque partners, s.r.o., 2014, ISBN 978-80-87923-01-6.

[6] Kadlec: *Agilní programování*. Brno: Computer Press, a.s., 2004, ISBN 80-251-0342-0.

[7] Kent, B.: *Extrémní programování*. Praha: Grada, a.s., 2002, ISBN 80-247-0300-9.

[8] Litera, T.: *Informační systém pro zpracování sportovních závodů v reálném čase*. Diplomová práce, Česká zemědělská univerzita, 2015.

[9] Maňák, J.: Jak můžeme zrychlit své webové aplikace? 2010-10-14, [online].
URL <http://www.manakmichal.cz/blog/optimalizace/jak-muzeme-zrychlit-sve-webove-aplikace/>

[10] Marek, M.: Průvodce prototypováním pro začátečníky. 2016-05-12, [online].
URL <https://www.interval.cz/clanky/pruvodce-prototypovanim-pro-zacatecniky/>

[11] Marian, B.: *Návrhové vzory v PHP*. Brno: Computer Press, a.s., 2012, ISBN 978-80-251-3338-5.

[12] Mark Nottingham, D. J.: Kešovací návod. 2004-01-01, [online].
URL <https://www.jakpsatweb.cz/clanky/caching-tutorial-czech-translation.html>

[13] Martin, P.: Nástroje pro analýzu rychlosti načtení stránky. 2017-02-10, [online].
URL <http://www.vzhurudolu.cz/prirucka/rychlost-nastroje>

[14] Mike, G.: *Z kodéra vývojářem - Nástroje a techniky pro opravdové programátory*. Brno: Computer Press, a.s., 2007, ISBN 978-80-251-1517-6.

[15] Peška, M.: Jak změřit načítání webu? A proč je důležitá pro SEO? 2014-03-18, [online].
URL <http://www.marketup.cz/cs/blog/jak-zmerit-rychlost-nacitani-webu-a-proc-je-dulezita-pro-seo>

[16] Ries: *Lean startup*. Brno: BizBooks, 2015, ISBN 978-80-265-0389-7.

[17] Sklář: *PHP5 - moduly, rozšíření a akcelerátory*. Brno: Computer Press, a.s., 2005, ISBN 80-86815-19-6.

[18] Vermilion: Responsive CSS Framework Comparison. 2015-10-07, [online].
URL <https://www.vermilion.com/responsive-comparison/>

[19] Ward, S. a. K.: *Building Web Solutions with the Rational Unified Process*. Rational, 1999.

[20] Yahoo: Best Practices for Speeding Up Your Web Site. 2016-11-20, [online].
URL <https://developer.yahoo.com/performance/rules.html>

8 Přílohy

Seznam příloh:

- Příloha A: Testovací přístupové údaje do systému

Příloha A

Testovací přístupové údaje do systému

Přihlášení do systému je možné na adrese http://reg3jez.skauting.cz/.

Testovací účty pro systém registrace

uživatelské jméno	**heslo**
admin@example.com	adminadmin
test@example.com	testtest

Testovací účty pro SkautIS

uživatelské jméno	**heslo**	**oprávnění**
kraj.vary	vary.Web2	Vedoucí/administrátor kraje
kraj.tgm	tgm.Web3	Vedoucí/administrátor kraje
okres.blansko	blansko.Web1	Vedoucí/administrátor okresu
stredisko.koprivnice	koprivnice.Web5	Vedoucí/administrátor střediska
snem.sneznik.kk	komise1	Člen kandidátní komise střediskového sněmu
snem.sneznik.uc	ucastnik1	Účastník střediskového sněmu