Jihočeská univerzita v Českých Budějovicích

Ekonomická fakulta

Současné trendy kódování webového frontendu

Diplomová práce

Bc. Roman Komrska

Vedoucí práce: doc. Ing. Ladislav Beránek, CSc.

České Budějovice 2016

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta ekonomická
Akademický rok: **2015/2016**

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Roman KOMRSKA**

Osobní číslo: **E14732**

Studijní program: **N6209 Systémové inženýrství a informatika**

Studijní obor: **Ekonomická informatika**

Název tématu: **Současné trendy kódování webového frontendu**

Zadávající katedra: **Katedra aplikované matematiky a informatiky**

Zásady pro vypracování:

Cílem práce je popsat a na praktickém příkladu ukázat současné praktiky kódování webových stránek. Výsledkem bude návrh a následné vytvoření funkční webové aplikace, která bude respektovat současné trendy kódování. Podrobně se práce bude zabývat otázkami jako je responzivita webu a dalšími body návrhu současného webu, a zejména problematice celkové optimalizace pro mobily, tablety a osobní počítače.

Metodický postup:

1. Studium odborné literatury.
2. Analýza a návrh aplikace, vývoj aplikace, testování.
3. Implementace do prostředí Internetu (nasazení na server).
4. Závěry a doporučení.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 - 50 stran**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

1. **Lubbers, P., Albers, B., & Salim, F. (2011).** *HTML5: programujeme moderní webové aplikace.* **Brno: Computer Press.**
2. **Goldstein, A., Lazaris, L., & Weyl, E. (2011).** *HTML5 a CSS3 pro webové designéry.* **Brno: Zoner Press.**
3. **Experti komunity jQuery. (2010).** *JQuery: Kuchařka programátora.* **Beijing: O'Reilly.**
4. **Galloway, J., Wilson, B., Allen, K., & Matson, D. (2014).** *Professional ASP.NET MVC 5.* **Indianapolis: John Wiley & Sons, Inc.**
5. **Řezáč, Jan. (2014).** *Web ostrý jako břitva.* **Jihlava: Baroque Partners.**

Vedoucí diplomové práce: **doc. Ing. Ladislav Beránek, CSc.**
Katedra aplikované matematiky a informatiky

Datum zadání diplomové práce: **15. března 2016**

Termín odevzdání diplomové práce: **15. září 2016**

doc. Ing. Ladislav Rolínek, Ph.D.
děkan

L.S.

prof. RNDr. Pavel Tlustý, CSc.
vedoucí katedry

V Českých Budějovicích dne 14. dubna 2016

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury. Prohlašuji, že v souladu s § 47 zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to – v nezkrácené podobě archivovaných Ekonomickou fakultou – elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

Datum Podpis studenta

Poděkování

Rád bych poděkoval panu Ing. Petru Hanzalovi Ph.D a panu doc. Ing. Ladislavu Beránkovi, za ochotu a cenné rady poskytované na konzultacích.

Obsah

1 Úvod

V současné době si může jen málokdo představit svět bez největší počítačové sítě, Internetu. Pod pojmem Internet si zřejmě většina lidí představí něco, kam se pomocí počítače připojí a může si velice jednoduše číst zprávy, komunikovat s lidmi z celého světa v reálném čase, sledovat videa a vlastně vyhledávat informace prakticky o čemkoliv. Ačkoli internet neslouží pouze pro výše zmíněné činnosti, tato diplomová práce se bude točit okolo této oblasti, tedy problematiky webových stránek, konkrétně webového frontendu.

Jak to vlastně všechno funguje? Pro představu je možné uvést situaci, kdy přijde člověk domů z práce a chce si přečíst, co nového se ve světě událo. Zapne tedy počítač, připojený k Internetu, otevře webový prohlížeč, do správné kolonky zadá požadovanou webovou adresu a zmáčkne tlačítko ENTER. Počítač v tu chvíli odešle požadavek na daný server, ten vše zpracuje a vrátí odesílateli požadavku vyžádanou odpověď, která se mu zobrazí ve webovém prohlížeči. V tuto chvíli je možné oddělit dva pojmy: backend a frontend. Procesem, odehrávajícím se na backendu je myšleno zmíněné zpracování požadavku na straně serveru. Frontend je oproti tomu to, co je odesláno do okna prohlížeče a v prohlížeči se vykonává. Jedná se o HTML kód, obohacený o CSS a JavaScript.

Tuto frontendovou část musel samozřejmě nějaký kodér vytvořit a navzdory tomu, že je frontend mnohdy zanedbáván, měla by se mu věnovat značná pozornost. Trendy v této oblasti se velice rychle mění a změnám je nutné se přizpůsobovat. Zde je možné zase ukázat příklad. Není to tak dávno, kdy bývalo domácí internetové připojení pomalé a proto bylo nutné šetřit při přenosu mezi klientem a serverem co nejvíce dat. Posléze docházelo k rapidním nárůstům rychlosti a zdálo se, že šetření dat už nebude tak důležité. V tom ale nastoupila éra smartphonů a jiných chytrých zařízení, která se začala připojovat k internetu. Zde kromě toho, že se opět objevil problém s rychlostí připojení (kvůli špatnému pokrytí mobilních operátorů) se vyskytl další problém a tím byla velikost obrazovek. Klasické webové stránky se na malých displayích špatně ovládaly, což vedlo k rozšiřování responzivního webdesignu. A to doposud nebyly zmíněny například problémy se zobrazováním stránek

v různých prohlížečích.

Pokud tedy majitel webových stránek chce spokojené návštěvníky napříč různými zařízeními, s různými velikostmi obrazovek a různými webovými prohlížeči, je potřeba, aby daný kodér věděl, jak se s možnými problémy vypořádat, aby byl výsledek opravdu kvalitní. Velikou výhodou je, že k této problematice existuje plno užitečných nástrojů ulehčujících práci a velká komunita nadšenců, kteří píší plno článků, točí videa a pořádají srazy s přednáškami na daná témata. Bohužel není mnoho zdrojů, kde by byly současné trendy kompletně shrnuty jako celek a tento nedostatek se pokusí tato diplomová práce napravit. V teoretické části bude shrnuto to nejdůležitější z teorie a následně v praktické části bude na názorných příkladech ukázáno, jakým způsobem se v dnešní době tvoří frontend webových stránek. Výsledkem bude funkční a responzivní webová aplikace, napsaná podle současných trendů.

1.1 Cíl práce

Cílem této diplomové práce je shrnout současné trendy kódování a na jejich základě vytvořit webovou aplikaci, která bude responzivní a použitelná na mobilních zařízeních. V teoretické části budou popsány novinky a současné trendy webového kódování a v praktické části podrobně rozepsán celý postup návrhu, vývoje, testování a následného nasazení na webový server tak, aby se čtenář, kterým může být například nováček v oboru, lépe zorientoval v dané problematice.

2 Metodika

Diplomová práce bude rozdělena na 2 části – na část teoretickou a praktickou. Aby bylo dosaženo kvalitního výsledku, je nutné dodržet určitou metodiku.

Úplně na začátku musí být studium dané problematiky. Jako zdroje je možné použít nejen doporučenou literaturu, ale i jiné vhodné zdroje. Ve chvíli kdy autor téma pochopí, zapíše své poznatky do teoretické části. V ideálním případě by se mělo jednat přesně o to, co se následně využije v praktické části.

K řešení praktické části dochází ve chvíli, kdy je hotová teoretická část a autor rozumí potřebné problematice. Jelikož výsledkem bude funkční responzivní webová aplikace, je potřeba si nejprve ujasnit, jak má tato aplikace vypadat. Sepíší se tedy všechny požadavky, na jejichž základě bude následovat tvorba prototypu nebo wireframů. Následovat bude nakódování frontendu a následně programování backendu, které celou aplikaci zprovozní. Celý postup, včetně vysvětlení jednotlivých kroků bude zaznamenán v praktické části a hlavní důraz bude kladen na úlohy, týkající se kódování frontendu.

3 Teoretická část

3.1 Responzivní webdesign

Responzivní webdesign byl původně definován tak, že za pomoci fluidního layoutu, obrázků a Media Queries vytvoří stránka, která se sama přizpůsobuje různým velikostem obrazovky. V současné době je ale potřba v tomto kontextu řešit větší množství úkolů, jako je například:

- potřeba zrychlit načítání na mobilních telefonech
- potřeba změnit pracovní postupy jako kódování PSD a vodopád
- potřeba občasné detekce zařízení na serveru
- responzivní obsahová strategie
- další techniky z oblasti mobilních telefonů, HTML5 atd. (Michálek, 2014a)

V souvislosti s responzivním webdesignem je dobré zmínit „Mobile First“. Tato fráze má mnoho významů, původně se jednalo o filozofii návrhu UI/UX webů a aplikací, ale může se také jednat o způsob psaní kódu (postupným vylepšováním) nebo designerské metodice. (Michálek, 2016b)

Mobile First neznamená upřednostňování mobilu před desktopem, ale vychází z myšlenky, že je dobré web/aplikci nejprve navrhnout na mobil a teprv následně na desktop. Důvodem je fakt, že na mobilu je podstatně omezenější prostor a návrháře donutí zaměřit pozornost na to nejdůležitější. (Michálek, 2016b)

3.2 Úvod do frontendu

3.2.1 HTML

Hypertext Markup Language (HTML) je značkovací jazyk pro tvorbu webových stránek, jehož autorem je Tim Berners-Lee a jehož historie sahá do konce osmdesátých let 20. století. Tim Berners-Lee v té době pracoval jako fyzik v Evropské organizaci pro jaderný výzkum a potřeboval nějakým efektivním způsobem sdílet dokumenty

přes Internet. Tehdy byl limitován možností použití pouze prostého textu ve spolupráci s technologiemi jako email, FTP a dalšími. Díky HTML bylo možné mít obsah uložený na centrálním serveru, ale s možností zobrazení skrze prohlížeč z lokálních počítačů. HTML vychází z jazyka SGML a jeho aktuální verze je HTML 5. (Mozilla Developer Network and individual contributors, 2016b)

Jak již bylo zmíněno výše, HTML je značkovací jazyk. Jazyk, který webovému prohlížeči říká, jak má danou stránku zobrazit – je možné pomocí něho oddělit obsahovou část od prezentační. HTML používá předdefinovanou sadu tagů (značek), kde každá značka definuje určitý druh obsahu. Ve zdrojovém kódu číslo 3.1 je vidět základní struktura stránky. Každá webová stránka, by měla mít na samém začátku umístěn doctype, který říká prohlížeči, o jakou verzi HTML (popřípadě XHTML) se jedná. Doctype uvedený v tomto příkladu říká, že se jedná o HTML5. Následuje tag html, který ohraničuje stránku a dělí se dále na head a body. V tagu head se píší například další nastavení, připojují CSS styly, javascriptové skripty atd. V tagu body je umístěn samotný obsah. V tomto příkladě se jedná o odstavec, obsahující text, kde jedna jeho část je následně zobrazena tučným písmem. HTML tagy mohou dále obsahovat atributy (v tomto případě „class“) a parametry (v tomto případě „odstavec“), které slouží pro další nastavení. (Mozilla Developer Network and individual contributors, 2016b)

Zdrojový kód 3.1: Základní struktura webové stránky v HTML5

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4       <title>Zde je umístěn titulek stránky</title>
5   </head>
6   <body>
7       <p class="odstavec">Zde se nachází odstavec a <
            strong>tento text je tučný</strong>.</p>
8   </body>
9 </html>
```

3.2.2 HTML5

HTML5 navazuje na HTML 4.01 a po ukončení vývoje XHTML 2 se jedná o hlavní cestu vývoje webů. Tvorba HTML5 začala v roce 2004 a podílely se na ní lidé z Apple, Mozilla Foundation a Opera Software. Přáli si rozvoj HTML a ne revoluci v podobě XHTML 2. V roce 2007 specifikaci převzalo W3C. (Sládek, 2010a)

HTML5 přichází s velkým počtem novinek. V následujících odstavcích jsou popsány ty nejznatelnější.

Sémantika

Dříve byla většina elementů na stránce umístěna v tagu , který ale nenese žádný sémantický význam a strojově s ním nelze pořádně pracovat. Toto dělení sloužilo především ke správnému stylování, ale nebylo možné se spolehnout na to, že autor bude například správně zadávat class atributy. Z tohoto důvodu vznikly nové sémantické tagy. (Sládek, 2010b)

Několik hlavních příkladů je uvedeno zde, ale existuje jich větší množství:

- header – tag, obalující hlavičku stránky
- footer – tag, obalující patičku stránky
- nav – tag, obalující hlavní navigaci (menu)
- section – tag, obalující danou sekci
- article – tag, obalující článek (Sládek, 2010c)

Video

Další výraznou novinkou je možnost vložení videa na web, aniž by bylo nutné používat externí plugin. HTML5 přináší nový element <video>, pomocí kterého lze video jednoduše do stránky vložit a v případě, že uživatelův prohlížeč tento tag nepodporuje, je možné nastavit fallback pro stažení videa. HTML5 také poskytuje API pro ovládání videa JavaScriptem. (Sládek, 2010d)

Audio

S přehráváním zvukových souborů na webové stránce to bylo podobné jako s videem. Také bylo pro přehrávání potřeba externích pluginů. V HTML5 slouží k přehrávání zvukových souborů na webové stránce element <audio>, který má podobně jako v případě videa své API, pomocí kterého lze ovládat JavaScriptem. (Malý, 2010a)

Formuláře

HTML5 přichází s novými možnostmi okolo webových formulářů. Dříve bylo možné pomocí HTML vyřešit jen základní věci a zbylé funkcionality bylo potřeba dotvářet například pomocí JavaScriptu. Asi hlavní novinkou je, že u formulářových prvků přibylo několik nových atributů a parametrů a s tím přišla i automatická validace zadaných dat. Je tedy například možné nastavit formulářové políčko na typ „email“ a prohlížeč nedovolí tento formulář odeslat, pokud nebude políčko vyplněno ve správném tvaru. Tato novinka je velice užitečná i v případě mobilních telefonů, protože dojde ke zobrazení klávesnice podle potřeby (pokud je například nastavený typ na číslo, zobrazí se klasická číslicová klávesnice). (Sládek, 2010e)

Data atributy

Dříve se často stávalo, že si vývojář ukládal určité informace do class atributů. S příchodem data atributů to už ale není třeba. V HTML5 je nyní možné psát atribut, který se skládá ze dvou částí:

- „data-“ – povinný prefix atributu
- „název“ – libovolný název atributu, který musí obsahovat alespoň 1 znak a neměl by obsahovat velká písmena (Malý, 2010b)

S hodnotami, které jsou uloženy v těchto atributech je následně možné pracovat pomocí JavaScriptu a to dvěma způsoby. Buď je možné jednoduše načíst element a z něho atribut „data-nazev“ nebo elegantnějším způsobem, pomocí vlastnosti dataset, na daném elementu. (Malý, 2010b)

3.2.3 CSS

Cascading Style Sheets (CSS) je stylovací jazyk, který se používá ve spolupráci s jazyky HTML nebo XML a je díky němu možné detailně určit, jak mají být dané elementy zobrazeny. CSS je jedním z klíčových jazyků pro web, je standardizován W3C specifikací a současnou verzí je CSS3. (Mozilla Developer Network and individual contributors, 2016a)

Zdrojový kód 3.2: Příklad CSS kódu

```
1 .odstavec{
```

```
2     color:blue;
3     font-family:Arial;
4     font-size:14px;
5     padding:10px
6 }
7 strong{
8     color:red
9 }
```

Zdrojový kód číslo 3.2 se dá použít společně se zdrojovým kódem číslo 3.1. Tag, který má u atributu „class“ nastaven parametr „odstavec“, bude obsahovat modré písmo, font bude nastaven na Arial o velikosti 14 pixelů a vnitřní vzdálenost textu od odkrajů bude 10px. Kromě toho bude tučný text červenou barvou.

Dále je samozřejmě nutné CSS kód nějakým způsobem do stránky připojit, čehož jde docílit pomocí pomocí těchto 3 způsobů:

- externě
- interně
- inline (Refsnes Data, 2016)

Externí vložení se provádí pomocí tagu <link>, vloženého do hlavičky stránky, viz. zdrojový kód 3.3, kde je odkázáno na externí CSS soubor „style.css“, který může obsahovat například zdrojový kód 3.2. (Refsnes Data, 2016)

Zdrojový kód 3.3: Externě vložený CSS kód

```
1 <head>
2 <link rel="stylesheet" href="style.css" type="text/css">
3 </head>
```

Interní vložení se provádí pomocí tagu <style>, který obsahuje CSS kód. Příklad je vidět na zdrojovém kódu 3.4. (Refsnes Data, 2016)

Zdrojový kód 3.4: Interně vložený CSS kód

```
1 <head>
2 <style>
3 /* zde může být vložen CSS kód */
```

```
</style>
</head>
```

Inline vložení se provádí pomocí atributu „style“, přičemž jako parametr je vložen samotný CSS kód. Příklad je vidět na zdrojovém kódu 3.5. (Refsnes Data, 2016)

Zdrojový kód 3.5: Inline vložený CSS kód

```
<p style="color:blue;font-family:Arial;font-size:14px;
   padding:10px">Text odstavce.</p>
```

3.2.4 CSS3

Nástupcem stylovacího jazyka CSS2 je pochopitelně CSS3, na kterém se pracovalo od roku 2005 a předchozí verzi rozšiřuje o nové možnosti a tím umožňuje kodérům elegantnější řešení jejich potřeb. (Šťastný & Šimeček, 2010)

Pomocí CSS3 lze nastavovat:

- stín textu
- způsob přetékání textu
- vlastní fonty
- vícesloupcovou sazbu textu
- míru roztažení pozadí
- pozici začátku pozadí
- velikost obrázku, umístěného na pozadí
- více obrázků, umístěných na pozadí
- barevné přechody
- RGBa barvy (barvy s průhledností)
- obrázek jako rámeček
- kulaté rohy
- stín elementu
- způsob, jak se počítá velikost boxu
- transformace tvarů objektů
- animace
- media Queries

- layout pomocí pružných boxů (Flexbox) (Michálek, 2015b)

Media queries

Media Queries jsou jedním z hlavních pilířů responzivního webdesignu. Lze pomocí nich nastavit styly pro rozsah šířky obrazovky, orientaci na výšku/šířku, poměr mezi CSS a hardwarovými pixely a poměr stran obrazovky. (Michálek, 2015b)

Jednotky

CSS3 přináší i nové jednotky:

- rem – obdoba rednotky „em“, ale s tím rozdílem, že rem se počítá podle root elementu, tedy od velikosti, nastavené na elementu <html>
- vh – procento z výšky viewportu (1vh je jedna setina výšky viewportu)
- vw – procento z šířky viewportu (1vw je jedna setina šířky viewportu)
- vmax – procento z delší strany viewportu (1vmax je jedna setina delší strany viewportu)
- vmin – procento z kratší strany viewportu (1vmin je jedna setina kratší strany viewportu) (Michálek, 2015b)

Flexbox

Flexbox je novou cestou pro tvorbu layoutu, zarovnávání a distribuci volné plochy (v češtině flex znamená slova přizpůsobivý a pružný). Lze pomocí něho tedy vytvořit pružné elementy, které dokáží vyplnit potřebný prostor, aniž by se muselo cokoli přepočítávat JavaScriptem. Používá se především pro tvorby layoutu komponent, ne celé stránky. Použití je relativně jednoduché. Vychází se z toho, že existuje nějaký flex kontejner, kterému se nastaví display na hodnotu flex a do něho spadají flex položky, což jsou všichni jeho přímí potomci. Hodně zjednodušeně by se dalo napsat, že zvládá to, co bylo možné vytvořit pomocí tabulkového layoutu, který se používal okolo roku 2000. To je sice pravda, ale pomocí flexboxu je možné toho udělat podstatně více. Zde je několik možných příkladů:

- všechny sloupce mohou být stejně vysoké
- možnost změny pořadí položek
- zarovnávání položek jak po ose x, tak y

- možnost nastavení zalamování položek
- možnosti zvětšování (nastavení, jak moc se může položka zvětšit)
- možnosti smršťování (nastavení, jak moc se může položka zmenšit)
- možnost nastavení výchozí velikosti položky
- možnost kombinovat jednotky (Michálek, 2015b)

3.2.5 JavaScript

Historie JavaScriptu sahá do roku 1995, kdy byla vydána jeho první verze v prohlížeči Netscape Navigator 2. V té době měl být používán pouze pro drobné doplnění dynamických prvků do webových stránek, jako je například validace nebo jednoduché obrázkové efekty. Poté, v Internet Exploreru 4, mu bylo umožněno sahat do objektového modelu stránek a díky tomu bylo možné pomocí JavaScriptu měnit libovolné části stránek. To dalo vzniku například vysouvacímu menu apod. Další revolucí byl příchod AJAXu, který umožnil komunikaci se serverem, bez nutnosti znovunačtení stránek. První velkou aplikací, která byla postavena převážně na JavaScriptu a AJAXu byl GMail. Dalším důležitým bodem je nástup javascriptových knihoven (příkladem může být jQuery), které usnadňují práci s DOM (Document Object Model) a odstraňují rozdíly mezi prohlížeči. To má ale za důsledek, že každá nová stránka je dnes obalena stovkami kilobyty javascriptového kódu. (Majda, 2008)

3.3 CSS preprocesory

CSS preprocesorem se myslí jazyk který je postaven nad CSS, do kterého přidává další vlastnosti nebo odstraňuje slabiny a následně se kompiluje co CSS. Soubory, obsahující tento kód mívají zpravidla specifickou koncovku, podle typu. Kompilace je možná těmito způsoby:

- speciální UI aplikace – aplikace, která lze snadno obsluhovat, ale má omezený rozsah funkcí
- editory a IDE – veliké množství editorů umí kompilovat preprocesory (Sublime, NetBeans, VisualStudio...)

- Grunt/Gulp – automatizátory procesů, které zvládají podstatně větší množství funkcí (budou rozebrány podrobněji dále)
- příkazová řádka – je potřeba jen vhodné prostředí (pro LESS Node.js, pro SASS Ruby...)
- kompilace v prohlížeči (Michálek, 2014b)

Nejpoužívanějšími preprocesory jsou LESS, SASS a Stylus. (Michálek, 2014b)

CSS preprocesory se svými vlastnostmi blíží imperativním programovacím jazykům, odstraňují nedostatky a rozšiřují možnosti psaní spravovatelného kódu. Zde jsou uvedeny hlavní technické vlastnosti:

- proměnné – jsou známy z imperativních programovacích jazyků, jako příklad využití se dá uvést možnost předeklarování proměnných z Bootstrapu
- mixiny – pojmenovaný kus kódu, který je možné opakovat na různých místech
- extend – vytváří kombinované selektory (oproti mixinům nekopíruje deklarace)
- zanořování
- zanořování media queries
- matematika – možnost použití matematických operací s čísly, jednotkami i barvami
- funkce pro práci s barvami
- podmínky (Stylus a SASS)
- cykly (Stylus a SASS)
- importy – oproti klasickému CSS nevytváří další request, ale vkládají importovaný soubor do zkompilovaného kódu (Michálek, 2014d)

3.3.1 LESS

Preprocesor LESS původně vytvořil v roce 2009 Alexis Sellier v Ruby a následně byl přepsán do JavaScriptu. V květnu 2012 se Alexis vzdal vedení vývoje, které předal týmu přispěvatelů, kteří nyní tento jazyk opravují a rozšiřují. (The core less team, 2016)

LESS se dá velice rychle naučit, protože plně respektuje deklarativní povahu CSS. V případě ale, že je potřeba tvořit cykly a podmínky, zápis už tak elegantní

není. Dá se tedy doporučit designerům a začátečníkům. (Michálek, 2014c)

3.3.2 SASS a Stylus

Autoři SASSu a Stylusu oproti autorům LESSu vycházeli z toho, že CSS jako takové je nedokonalé a je potřeba ho upravit tak, aby se podobalo běžným programovacím jazykům. Oba proto mají více imperativní charakter – syntaxe podmínek a cyklům se podobá například jazyku PHP. Problém ale může nastat ve chvíli, když si neprogramátor čte SASS kód. Oproti LESSu bude podstatně horší mu porozumět. SASS a Stylus by se tedy daly doporučit vývojářům. (Michálek, 2014c)

3.4 Javascriptové knihovny a frameworky

Jak již bylo uvedeno výše, pro usnadnění si práce s JavaScriptem se dají použít různé knihovny a frameworky. Mezi nejznámější patří například jQuery, React.js nebo Angular.

3.4.1 jQuery

Knihovnu jQuery vydal v roce 2006 John Resig. Velikou výhodou bylo, že pomocí této knihovny dokázali vývojáři vytvořit dříve složité úlohy velice snadno a rychle, díky velmi jednoduché práci s DHTML (dynamické HTML), práci s HTML dokumenty, zpracovávání událostí prohlížeče, animacím, práci s AJAXem a psaní skriptů, které fungují napříč prohlížeči. (Experti komunity jQuery, 2010)

Filozofii knihovny jQuery je možné shrnout do těchto bodů:

- Hledání elementů a manipulace s nimi.
- Řetězené volání metod na skupině elementů.
- Používání obalu jQuery a implicitní iterace. (Experti komunity jQuery, 2010)

3.4.2 React

Další javascriptovou knihovnou je React, který byl uvolněn Facebookem v květnu 2013. V prvních chvílích moc úspěchu nesklidil, ale postupem času se ukázalo, že jde jen o nepochopení celé knihovny. Nejzřejmější výhodou Reactu je, že vývojáře oddělí od práce s DOMem. V komponentách pouze popíše, jak má výsledek vypadat, React si poskládá vlastní (virtuální) DOM a ten následně porovnává se skutečným DOMem a podle toho jej upravuje. To je veliký rozdíl oproti jQuery, ve kterém vývojář pracuje s jednotlivými DOM elementy. (Mikšů, 2016)

Jednou z výhod Reactu oproti jQuery je možnost uchovávání stavů aplikace a další práce s nimi. Jako příklad je možné uvést počet položek v nákupním košíku, který se pomocí JavaScriptu aktualizuje. Tento počet je možné použít na víc místech aplikace, což by v jQuery znamenalo aktualizovat několik DOM elementů. Dále vývojáři Reactu vytvořili JSX nadstavbu nad JavaScriptem, díky které je možné zapisovat HTML do JavaScriptu a tím velice zjednodušit celkový kód. (Steigerwald, 2014)

3.4.3 Angular

Velice oblíbený je také javascriptový framework Angular, vyvíjený Googlem. Angular vychází z modelu MVC a je možné jej kombinovat s dalšími javascriptovými knihovnami, jako je například jQuery. Nejčastěji uváděnými výhodami jsou Two Way Data-Binding, Dependency Injection, testovatelnost, direktivy a znovupoužitelnost komponent. (Mrozek, 2012)

Bohužel Angular není bezchybný a na Internetu se objevuje mnoho kritických článků. Mezi hlavní nevýhody patří výkon a ne příliš dobré navržení. Tyto problémy by ale měly být vyřešeny v Angularu 2. (Steigerwald, 2015)

3.5 Node.js

Node.js je javascriptové prostředí, které je velice výkonné, rychlé a událostmi řízené. Základem je javascriptový interpret V8, který pochází z Google Chrome a nad ním je

C++ vrstva. Node.js je navržen tak, aby řešil nízkoúrovňové problémy za uživatele a jeho prostředím je čistý JavaScript v jediném vlákně. S Node.js úzce souvisí také balíčkovací systém NPM, který umožňuje jednoduché prohlížení a instalaci svých aplikací, včetně závislostí. (Nešetřil, 2010)

Na frontendu se Node.js využívá především kvůli usnadnění vývoje a správě vývojářských závislostí. Jako příklady používaných nástrojů lze uvést:

- Node.js – aplikace, která umožňuje běh JavaScriptu na příkazové řádce
- NPM – javascriptový balíčkovací systém
- Bower – správce frontendových balíčků
- Grunt/Gulp – nástroj pro běh vývojářkých úloh
- Grunt/Gulp pluginy – jednotlivé úlohy Gruntu/Gulpu, které zajišťují například minifikace souborů nebo kompilace preprocesorů do čistého CSS (Michálek, 2016d)

3.5.1 Grunt a Gulp

Aplikaci Grunt je možné nazvat „The JavaScript Task Runner“ a slouží webovým vývojářům k automatizaci nudných, opakujících se úkolů. Používá se způsobem, že se spustí úloha na příkazové řádce, což samo začne hlídat změny v souborech a podle toho provádět požadované operace. Jako příklad je možné uvést minifikaci CSS a JS souborů, jejich spojování do jednoho kvůli zlepšení rychlosti, kompilace CoffeeScriptu do JavaScriptu a stovky dalších úloh, přičemž není problém napsat vlastní. Malou nevýhodou je, že uživatel Gruntu musí alespoň základním způsobem umět používat příkazovou řádku. Existují ale i GUI alternativy (CodeKit, Prepros, Koala), které ale neumí tolik úloh. Grunt se konfiguruje pomocí souboru Gruntfile.js a výhodou je jeho verzování, tudíž je ho možné krásně sdílet s celým týmem. (Michálek, 2016c)

Gulp je aplikace stejného typu, jako Grunt. Vznikl ale s cílem vše zjednodušit a zrychlit. Gulp masivně využívá Node.js streamy a pomocí pipeline se předávají data mezi úlohami. Na rozhodnutí, jestli zvolit Grunt nebo Gulp moc nezáleží, jde především o to, co bude komu individuálně vyhovovat. Grunt je starší, má více pluginů a je používanější. U Gulpu je oproti tomu uváděna vyšší rychlost. (Ožana, 2014)

Pluginy

Zde je uvedeno několik často používaných Grunt pluginů:

- grunt-autoprefixer – doplnění CSS3 prefixů
- grunt-pixrem – tvorba px fallbacků k rem
- grunt-criticalcss – plugin, který dokáže generovat critical CSS
- grunt-contrib-cssmin – minifikace CSS souborů
- grunt-uncss – plugin, který odstraňuje nepoužívané CSS
- contrib-less, contrib-sass, contrib-stylus – kompilace preprocesorů LESS, SASS a Stylus
- grunt-styleguide – generování dokumentace
- grunt-contrib-concat – spojení JS souborů do jednoho
- grunt-contrib-uglify – minifikace JS souborů
- grunt-contrib-htmlmin – minifikace HTML
- grunt-svg2png – převod SVG do PNG
- grunt-responsive-images – generování zmenšenin obrázků pro responzivní weby
- grunt-contrib-watch – hlídání změn souborů
- grunt-browser-sync – živý reload stránky a synchronizace s dalšími zařízeními
- grunt-ftp-deploy – upload projektu přes FTP na server
- jit-grunt – zrychlení náčítání Grunt pluginů
- grunt-concurrent – plugin, umožňující běh více pluginů najednou (Michálek, 2016a)

Ty samé nebo podobné pluginy jsou dostupné také pro Gulp. Většinou mají i stejný název, popřípadě stačí v názvu vyměnit slovo „grunt“ za „gulp“.

3.5.2 Yeoman

Yeoman je nástroj, který pomáhá při zakládání nových projektů. Jedná se o generátorový ekosystém, který dokáže podle nastavení vytvořit kostru projektu, včetně všech užitečných funkcí, jak daný vývojář požaduje. Oficiálně je poskytováno „Yeoman workflow“, což je balíček, obsahující nástroje a frameworky, které pomáhají vývojářům při tvorbě aplikací. Yeoman workflow obsahuje tyto 3 typy nástrojů,

zlepšujících produktivitu při tvorbě webu:

- scaffoldingový nástroj (Yeoman)
- buildovací nástroj (Grunt, Gulp...)
- správce balíčků (NPM, Bower...) (The Yeoman Team, 2016)

Yeoman staví „lešení“ nové aplikace – do konfiguračního souboru (například gruntfile.js) vkládá buildovací úlohy a závislosti, buildovací nástroj se používá pro sestavování, náhled a testování projektu a správce balíčků řeší dotažení závislostí. Všechny 3 části jsou vyvíjeny odděleně, což ale nebrání tomu, aby společně fungovaly výborně. (The Yeoman Team, 2016)

3.6 Verzovací systémy

Historie verzovacích systémů se začíná psát na přelomu 70. a 80. let a v dnešní době jsou velice významným pomocníkem téměř každého vývojáře. Používá se jako uložiště zdrojových kódů a dalších souborů, udržuje jejich historii a je dále možná spolupráce s dalšími vývojáři. Zde je seznam hlavních výhod verzovacích systémů:

- Uchovávání historie verzí – Ve chvíli, kdy vývojář zjistí, že současná verze oproti té předchozí nefunguje a nemůže přijít na to, kde nastala chyba, je velice snadné se vrátit k předchozí funkční verzi a následně porovnávat změny, které byly udělány. Ruční verzování, dělané pomocí kopírování projektů do očíslovaných adresářů je už v dnešní době přežitek.
- Týmová spolupráce – Pomocí verzovacího systému je velice snadné sdílení projektů, na kterých pracuje více lidí – soubory si vzájemně nepřepisují.
- Práce ve více větvích – hodí se při vývojí softwaru a umožňuje paralelní vývoj několika různých funkcionalit. (Kučera, 2011)

3.6.1 Git

Git je dnes asi nejznámějším verzovacím systémem a byl vytvořen v roce 2005 Linusem Torvaldsem a komunitou vývojářů jádra Linuxu. Mezi hlavní požadované vlastnosti patřilo:

- rychlost
- jednoduchost
- silná podpora paralelního vývoje
- plná distribuovatelnost
- schopnost efektivně spravovat vélké projekty (jako je například jádro Linuxu) (Chacon, 2009)

Od roku 2005 Git vyzrál v použitelný systém a i dnes si uchovává své kvality. Je extrémě rychlý, pracuje se s ním efektivně a i pro velké projekty nabízí kvalitní systém větvení. (Chacon, 2009)

3.7 Frontendové frameworky

Frontendovým frameworkem se myslí sada HTML, CSS a JS komponent, které se dají následně používat pro tvorbu webového frontendu, kde může kodérovi usnadnit práci, zlepšit funkčnost a výkon webu nebo řešit problematiku zobrazování na displayích různých velikostí. Asi nejznámější frontendovým frameworkem je Bootstrap, ale existuje jich veliké množství, jako například Foundation, Semantic UI, Pure!, HTML KickStar a další. (Fichtner, n.d.)

3.7.1 Bootstrap

Bootstrap byl původně vytvořen vývojáři z Twitteru v roce 2010 a je nejpopulárnějším HTML, CSS a JS frameworkem na světě, pro vývoj responzivních, mobile first webových projektů. Finální verze je dostupná v čistém CSS, ale jeho zdrojové kódy lze stáhnout i v LESS a SASS, díky čemuž si lze snadno vytáhnout jen určité části a ty následně použít ve svém projektu. Jako jedna z nejužitečnějších komponent se dá použít mřížka, díky které lze snadno tvořit weby optimalizované pro mobily, tablety i stolní počítače. Kromě toho ale obsahuje veliké množství HTML, CSS a JS komponent, které lze velice snadno používat a mají i hezký design. To celé je velice pěkně zdokumentováno na oficiálních stránkách. (Otto, 2016)

3.7.2 Foundation

Foundation je asi druhým nejznámějším frontendovým frameworkem, který je vyvíjen firmou ZURB, Inc. (ZURB, Inc., 2016)

Ve své hlavní podstatě slouží k tomu samému, jako Boostrap, akorát se v určitých funkcionalitách liší. Příklady jsou uvedeny níže. (Ševčík, 2014)

Boostrap (verze 3)

- mřížka má 4 velikosti (malý, střední, velký, extra velký)
- veliké množství UI elementů
- jako jednotky používá pixely
- používá preprocesory LESS a SASS. (Ševčík, 2014)

Foundation (verze 5)

- mřížka má 3 velikosti (malý, střední, velký)
- menší množství UI elementů
- jako jednotky používá root em (rem)
- používá pouze preprocesor SASS (Ševčík, 2014)

3.8 Rychlost načítání

S příchodem mobilních zařízení připojených do internetu začalo být velice důležité řešit rychlost načítání. Důvodem je to, že ne každý člověk s mobilním telefonem není odkázán na 2G. Navíc rychlost načítání začal řešit i vyhledávač Google a stránky podle toho zohledňuje. (Sládek, 2015)

Zde je několik bodů, jakým způsobem se dá rychlost načítání značně ovlivnit:

- zvážit nutnost použití různých knihoven pluginů a případně je nepoužít (mnohdy je možné funkcionalitu napsat na pár řádků)
- slučovat JavaScripty a CSS
- používat obrázkové sprity
- načítat zdroje z různých domén

- používat správně obrázky (komprese, SVG nebo ikon fonty, base 64, progresivní JPEG, minifikace atd.)
- minifikace HTML, CSS, JS a SVG
- používat co nejméně řezů fontů
- vhodně nastavit server (expires header, gzip, cache)
- použití serverové detekce zařízení
- vyvarovat se častým chybám (prázdné src a href atributy, @import ve stylech, u obrázků nastavovat šířku a výšku přímo na elementu a ne přes CSS, načítat CSS a JS ze správných míst atd.) (Krejčí, 2015)

3.9 Rychlost zobrazení a Critical CSS

Od rychlosti načítání je potřeba odlišit rychlost zobrazení. Rychlost zobrazení by se dala vysvětlit takto: Klient odešle požadavek na zobrazení dané stránky. Od serveru obdrží odpověď v podobě HTML kódu, který prohlížeč začne postupně zpracovávat. Prohlížeč tedy obdržel HTML, které následně prochází a zjišťuje, že musí dále načíst CSS, JavaScript a další soubory. Většinou prohlížeč stránku nezobrazí do doby, než bude mít načten celý CSS soubor. (Michálek, 2015a)

Aby se tedy zrychlilo načítání, používají se takzvané critical CSS (styly, které jsou nutné pro zobrazení stránky na určitém rozlišení). Ty se dají generovat například pomocí Gruntu a vkládají se inline do hlavičky. Zbylé CSS se načítá až po zobrazení stránky asynchronně. Tento proces může načítání rapidně zrychlit na pomalém připojení až o několik sekund. (Michálek, 2015a)

3.10 Užitečné nástroje

Sublime Text

Sublime Text se dá použít jako ideální vývojové prostředí. Je rychlý, v základu sice moc funkcí nemá, ale jde rozšířit o veliké množství pluginů, díky kterým si kodér dokáže práci zpříjemnit. Bohužel je potřeba věnovat určitý počáteční čas nastavení. Zde je uvedeno několik příkladů pluginů a čeho lze pomocí nich

dosáhnout:

- Emmet – obrovské zrychlení psaní kódu za pomoci zkratek
- Grunt – spouštění Gruntu přímo z editoru
- SublimeGit – umožňuje práci s Gitem přímo z editoru
- Alignment – zarovnává textové bloky na tabulátory
- Sublime-HTMLPrettify – správně odsadí rozpadlé HTML
- HTML5, jQuery, LESS a další – zvýrazňování syntaxe (Michálek, 2016e)

Samozřejmě tento editor nemusí vyhovovat každému, ale existuje množství alternativ (zde by mohlo být dobré zmínit Brackets, což je editor od Adobe, který umožňuje pracovat s PSD soubory). Pokud někdo vyhledává robustnější IDE, může využít například NetBeans nebo PHPStorm. (Michálek, 2016e)

BrowserStack

BrowserStack je velice užitečný nástroj pro testování webu v reálném čase, na různých zařízeních. Obrovskou výhodou je snadné použití a skutečnost, že stránka je v mnoha případech testována na fyzických strojích a ne na emulátoru. Pro debugování lze použít různé nástroje, jako například Chrome Developer Tools. Nevýhodou cena, která se platí formou měsíčního paušálu a někomu se může zdát vysoká. (BrowserStack Limited, 2016)

Adobe® Photoshop®

Adobe® Photoshop® je software primárně určený primárně na úpravu fotografií, který se hodně rozšířil i ve webdesignu, kde se pomocí něho tvoří grafické návrhy. Díky různým pluginům je ho možné rozšiřovat o další funkcionality.

Avocode

Avocode je software, usnadňující spolupráci designerů a vývojářů (Avocode, Inc., 2016a). Je možné pomocí něho design sdílet, psát k němu poznámky, řadit do kategorií atd. Kodérovi usnadní práci například tím, že pomocí něho dokáže snadno, pomocí pár kliknutí vytahávat obrázky a dokonce generovat části CSS kódu. (Avocode, Inc., 2016b)

4 Praktická část

V praktické části této diplomové práce bude na praktickém příkladu ukázáno, jakým způsobem se dnes kódují webové stránky. Jako příklad bude použita webová aplikace pro evidenci osobních výdajů, která byla vyvinuta přímo pro tento účel. Tato část bude rozdělena do následujících sekcí:

- popis aplikace (definování, jak má výsledná aplikace vypadat a co bude možné pomocí ní dělat)
- návrh aplikace (tvorba wireframů a prototypů)
- rozbor zdrojového kódu (hlavní část, kde bude podrobně rozebrán zdrojový kód)

4.1 Popis aplikace

Jak již bylo zmíněno výše, aplikace bude sloužit pro evidenci osobních výdajů a jelikož by se případným koncovým uživatelům mohla hodit kdekoli na cestách, bude nutné, aby byla optimalizována pro mobilní zařízení a aby šla používat ideálně i na pomalém 2G připojení.

Potřebné budou tedy následující funkcionality:

- Registrace uživatelů
- Přihlašování uživatelů
- Práce s kategoriemi nákladů
    - Přidávání
    - Editace
    - Mazání
    - Zobrazení
- Práce se samostatnými náklady
    - Přidávání
    - Editace
    - Mazání
    - Zobrazení

- Zobrazení informací o aplikaci
- Možnost kontaktovat autora
- Možnost změny hesla
- Odhlášení uživatele

4.2 Návrh aplikace

4.2.1 URL adresy

Výše byly zmíněny požadavky na funkcionality. Na jejich základě byly navrženy následující adresy. Jako doména je použita „dp.romankomrska.cz“, protože zde bude do data obhajoby této diplomové práce aplikace dostupná.

- http://dp.romankomrska.cz/prihlaseni – stránka pro přihlašování uživatelů
- http://dp.romankomrska.cz/registrace – stránka pro registraci uživatelů
- http://dp.romankomrska.cz/ – hlavní stránka aplikace, kde bude možné evidovat náklady
- http://dp.romankomrska.cz/kategorie – stránka, kde bude možné nastavovat kategorie nákladů
- http://dp.romankomrska.cz/o-aplikaci – stránka, kde bude možné zobrazit informace o aplikaci
- http://dp.romankomrska.cz/kontakt – stránka s kontaktními údaji a formulářem
- http://dp.romankomrska.cz/muj-ucet – stránka s informacemi o účtu a možností změnit heslo
- http://dp.romankomrska.cz/odhlasit – adresa pro odhlášení

4.2.2 Wireframy a detailní popis jednotlivých pohledů

V tuto chvíli, když jsou známy jednotlivé URL adresy, se může přejít k návrhu jednotlivých pohledů. K tomuto účelu bude využit program Axure RP 8 od firmy Axure Software Solutions, Inc. Výhodou tohoto softwaru je, že je práce s ním velice intuitivní a kromě wireframů nabízí také možnost tvorby prototypů, čímž je

myšlena ve své podstatě sada wireframů, které mohou být vzájemně propojeny a je tedy možné pomocí nich vytvořit jakýsi funkční model. V tomto případě ale bohatě postačí wireframy s výstižný popisem. Všechny stránky budou navrhovány metodou „mobile-first“, tedy metodou, že se začíná návrhem mobilní verze, která se následně rozšiřuje o prvky verze pro tablet a počítač.

Přihlášení a registrace

Pro tyto 2 stránky nebude potřeba tvořit wireframe, ale postačí základní popis. Obě stránky budou mít v horní části název aplikace (pro tyto účely MoneyTrack) a na prostředku zobrazen formulář. Nad formulářem se budou v případě potřeby zobrazovat hlášky například úspěšné registraci apod. Pro přihlašovací stránku bude formulář následující:

- políčko pro email
- políčko pro heslo
- tlačítko pro přihlášení
- tlačítko, odkazující na registraci

Registrace bude vypadat následujícím způsobem:

- políčko pro zadání emailu
- políčko pro heslo
- políčko pro zopakování hesla
- tlačítko pro registraci

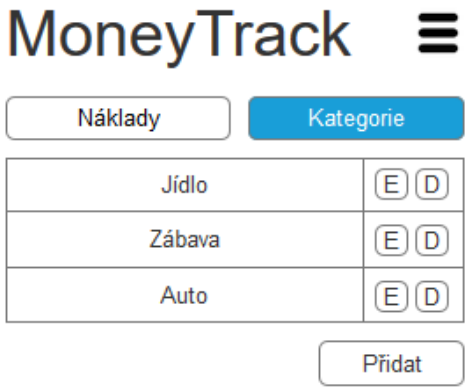
Kategorie

Na této stránce bude možné evidovat kategorie. Tato stránka bude mít hlavičku stejnou jako všechny další stránky a bude se skládat z názvu aplikace, menu a druhého menu. Samotný specifický obsah této stránky bude tvořit blok s tabulkou, ve které bude možné přidávat, editovat a mazat kategorie. Pro editaci a přidávání bude vyskakovat formulář, ve kterém se bude vyplňovat název a popis. Wireframe pro mobil je znázorněn na obrázku 4.1 a pro desktop na obrázku 4.2.

Na těchto obrázcích je vidět jak se navrhovaná aplikace bude přizpůsobovat ve-

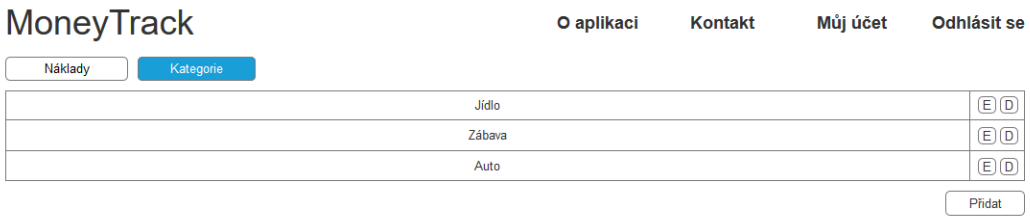
likostem displaye. Tlačítko s písmenem E bude zobrazovat editovací formulář a tlačítko D bude zobrazovat dotaz, zda chceme opravdu obrázek smazat.

Obrázek 4.1: Wireframe stránky „Kategorie“ pro mobil



Zdroj: vlastní zpracování

Obrázek 4.2: Wireframe stránky „Kategorie“ pro počítač

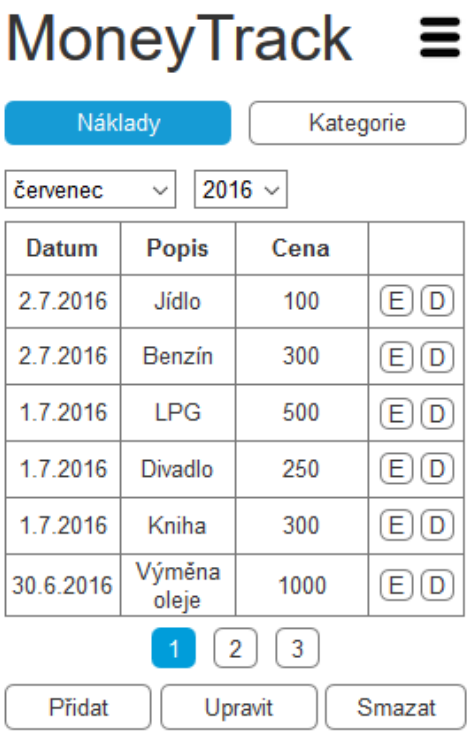


Zdroj: vlastní zpracování

Hlavní stránka (přehled)

Hlavní stránka bude obsahovat kromě hlavičky tabulku se stránkováním. Tabulka bude v principu velice podobná té s kategoriemi, akorát s tím rozdílem, že zde bude přidané také stránkování. Nad tabulkou budou umístěny 2 selectboxy, pomocí kterých bude možné vybírat měsíc a rok. Wireframe pro mobil je na obrázku 4.3 a pro počítač na obrázku 4.4. Zde je vidět, že mobilní verze obsahuje pouze to nejdůležitější (tabulku), zatímco verze pro větší monitory je rozšířena o graf.

Obrázek 4.3: Wireframe hlavní stránky pro mobil



Zdroj: vlastní zpracování

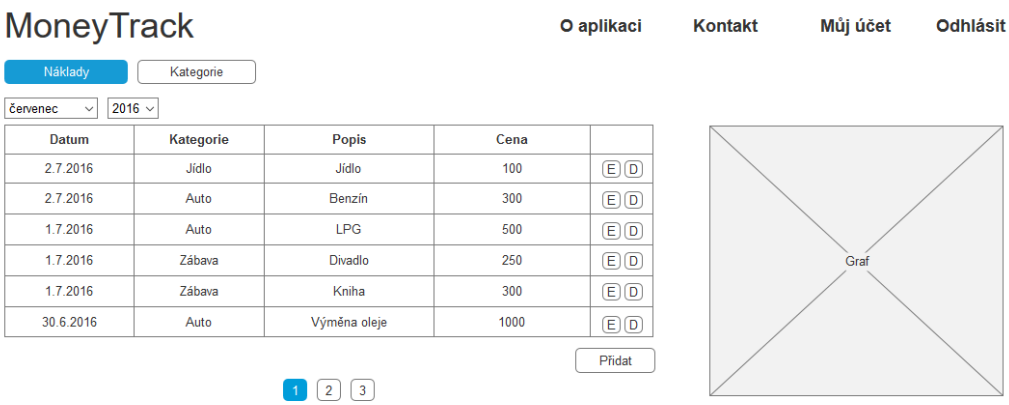
Ostatní stránky

Ty stránky, pro které bylo dobré ukázat wireframy již byly zmíněny a dále budou popsány ty jednodušší.

První z nich je stránka, zobrazující informace o aplikaci. Zde budou umístěny základní informace, proložené nějakým vhodným obrázkem. Samozřejmě se bude muset zajistit, aby se tato stránka zobrazovala správně na všech zařízeních.

Druhou je stránka s kontaktními údaji a kontaktním formulářem. Ta se bude skládat ze dvou bloků. Jeden bude obsahovat kontaktní formulář a druhý kontaktní údaje. Specialitou je to, že tyto bloky se budou na mobilu zobrazovat pod sebou, zatímco na dalších zařízeních vedle sebe. Na mobilech se zobrazí nejprve kontaktní údaje a pod nimi kontaktní formulář, ale na ostatních zařízeních bude nalevo kontaktní formulář a kontaktní údaje napravo. Bude tedy nutné zajistit prohození bloků podle velikosti displaye.

Obrázek 4.4: Wireframe hlavní stránky pro počítač



Zdroj: vlastní zpracování

4.3 Technické řešení

4.3.1 Potřebné technologie

V tuto chvíli, kdy je aplikace navržena, je možné přejít ke konkrétnímu technickému řešení. Pro frontendovou část aplikace budou použity jazyky HTML5, CSS3 a JavaScript, pro backendovou část jazyk PHP. Tyto technologie budou doplněny o následující:

- LESS – CSS preprocesor, usnadňující psaní CSS kódu
- Boostrap – frontendový framework, ze kterého se použijí jen některé části
- React – javascriptová knihovna pro tvorbu uživatelských rozhraní
- jQuery – knihovna, usnadňující zápis JavaScriptu a AJAXových požadavků

4.3.2 Potřebné nástroje

Aby byl vývoj efektivní, je nutné pracovat se správnými nástroji, jinak se může i několikanásobně prodloužit. V této sekci budou představeny nástroje, pomocí nich byla aplikace vytvořena a které mnohdy vývoj urychlily.

Prvním takovým nástrojem je software Sublime Text 3. Jedná se o velice rychlý textový editor, oblíbený velkým množstvím webových vývojářů a který je možné

rozšířit o obrovské množství pluginů a každý si určitě najde to své, co zrovna potřebuje. Bohužel je ale nutné zmínit, že začínající vývojář stráví velké množství času, než si tento software vyladí ke své dokonalosti. Zde je seznam některých použitých pluginů:

- AutoFileName – doplňování cest k souborům
- Color Highlighter – znázorňování barvy
- Emmet – plugin, urychlující psaní kódu
- HTML5 – podpora HTML 5 syntaxe
- LESS – podpora LESS syntaxe

Dalším editorem, který bude použit pro vývoj, je NetBeans. Ten bude využit pro tvorbu backendu, kvůli dobré podpoře Nette Frameworku a tomu, že je dostupný zdarma. Bohužel se jedné o robustní řešení a je oproti Sublime Textu pomalý.

Další nástroje:

- Xampp – balík, umožňující snadno spustit lokální webový server
- WinSCP – FTP klient, který dokáže hlídat změny v adresáři a okamžitě je nahrávat na server
- Node.js – prostředí, které umožňuje instalovat veliké množství užitečných pluginů (například Gulp a Bower)
- Composer – nástroj pro řízení závislostí v PHP
- GIT – verzovací systém

4.3.3 Adresářová struktura

Když se vyvíjí webové aplikace, je dobré oddělit vývoj frontendu od backendu. Většinou se tedy nezačíná naráz programovat backend a do něho rovnou nasazovat a ladit frontend, ale nejprve se vyvine a odladí frontendová část, která se posléze začne nasazovat do backendové části.

Vzhledem k tomu, že se v dnešní době využívá veliké množství externích knihoven, které jsou závislé na dalších knihovnách a některé z nich mají vlastní úpravy jazyků, je potřeba oddělit vývojovou část od produkční, protože při vývoji dochází ke kompilaci zdrojového kódu a není tedy dobré míchat vše v jednom adresáři. O kom-

pilaci se většinou stará nějaký javascriptový task runner ve spolupráci se svými pluginy. Adresářová strukturá může být taková, jak každému jednotlivci vyhovuje. V tomto pojektu bude použita následující:

- dev – složka pro vývoj
  - assets – složka, obsahující moduly nainstalované Bowerem
  - css – překompilované css soubory
  - fonts – složka s fonty
  - images – složka s nekomprimovanými obrázky
  - js – vývojové verze javascriptových souborů
  - less – vývojové verze less souborů
  - node_modules – složka s pluginy z Node.js ekosystému
- prod – složka, obsahující PHP soubory a jediný překompilovaný CSS soubor
  - fonts – stejné fonty jako ve vývojové verzi
  - js – překompilované javascriptové soubory

4.3.4 Vývoj frontendové části

Frontendová část byla v tomto projektu vyvinuta jako první. Veškeré soubory byly umístěny v jednotlivých adresářích ve složce „dev“ přesně tak, jak bylo zmíněno výše a za pomoci Gulpu a jeho pluginů bylo vše kompilováno do produkční verze. Ve chvíli, kdy byla frontendová část odladěna, začala se nasazovat na backend. Všechny tyto principy budou následně ukázány společně s backendovou částí. Dojde tak k lepšímu pochopení celého fungování aplikace a nebudou určité kroky popisovány dvakrát.

4.3.5 Rozbor zdrojového kódu

V této části bude podrobně rozebrán zdrojový kód jednotlivých souborů. Vysvětlováno bude na hotové aplikaci, naprogramované na Nette Frameworku. Na zdrojovém kódu frontendu se v principu nemění nic, jen je pouze dosazen do Latte šablon. Trošku odlišné bude nastavení Gulpu, ale to bude následně dovysvětleno. V této části navíc nebudou kompletní zdrojové kódy, ale jen jejich podstatné části. V případě

zájmu je možné je nalézt v přílohách a nebo na přiloženém CD.

Nette Framework

Nette Framework je v České republice nejoblíbenější backendový MVC framework a tato aplikace je na něm postavena. Velice snadno se pomocí něho dala vytvořit kostra této aplikace, přičemž odpadly různé rutinní činnosti. Jelikož se tato práce nemá backendem zabývat, budou zde zmíněny pouze ty části, které jsou nutné pro pochopení potřebných funkcionalit. Jako první je třeba zmínit soubor RouterFactory.php, který se nachází v adresáři app/router a jeho hlavní metoda, ve které jsou nastaveny URL adresy se zobrazena ve zdrojovém kódu 4.1. V tomto souboru je dobré si všimnout konstruktoru třídy Route, které se předávají jako parametry požadované URL a presenter s pohledem. Presenter značí vrstvu „C“ v MVC modelu a pohled „V“. V tomto případě se bude daný pohled renderovat podle toho, co do něho odešle presenter.

Zdrojový kód 4.1: RouterFactory.php

```
1  public static function createRouter() {
2    $router = new RouteList;
3    $router[] = new Route('/prihlaseni', 'Sign:login');
4    $router[] = new Route('/registrace', 'Sign:registration
       ');
5    $router[] = new Route('', 'Dashboard:default');
6    $router[] = new Route('kategorie', 'Categories:default')
       ;
7    $router[] = new Route('o-aplikaci', 'About:default');
8    $router[] = new Route('kontakt', 'Contact:default');
9    $router[] = new Route('muj-ucet', 'Accounts:default');
10   $router[] = new Route('odhlasit', 'Sign:out');
11   return $router;
12 }
```

Přihlášení

Jednotlivé stránky budou vysvětlovány v takovém pořadí, jak se s nimi uživatel setká, když aplikaci poprvé zapne. Jako první stránku uživatel tedy uvidí přihlášení. Tato stránka spadá do presenteru „Sign“.

Zdrojový kód 4.2: Metoda pro vytvoření komponenty přihlašovacího formuláře

```
1  public function createComponentLoginForm() {
2    $form = new Form;
3    $form->addText('email')
4         ->setType('email')
5         ->setRequired('Vyplňte e-mail.')
6         ->setAttribute('placeholder', 'jan@novak.cz');
7    $form->addPassword('password')
8         ->setRequired('Vyplňte heslo.')
9         ->setAttribute('placeholder', 'heslo');
10   $form->addSubmit('send', 'Přihlásit');
11   $form->onSuccess[] = array($this, 'loginFormSucceeded');
12   return $form;
13 }
```

Ve zdrojovém kódu 4.2 je vidět, jakým způsobem se v Nette vytváří komponenty. Vše se točí okolo objektu „Form“, do kterého se přidávají jednotlivé položky. V tomto případě se přidává textové pole pro zadání emailu, které je povinné a při nevyplnění se zobrazí chybová hláška. Dále se nastavuje placeholder, což je předvyplněný text, který po kliknutí do pole zmizí. Pole pro zadání hesla je podobné.

V tuto chvíli je možné přejít k frontendové části této stránky, která je zobrazena ve zdrojovém kódu 4.3. Jak je vidět hned z prvního řádku, jako hlavní šablona, do které je tato posléze vložena se načítá ze souboru „@layout2.latte“. Následně je vidět využití frameworku Bootstrap. Třída „alert“ nastavuje styly pro vypsané hlášky (pokud existují). Většinou se použije například „alert-danger“, „alert-success“ nebo nějaké další, které Bootstrap obsahuje.

Dále jsou ve zdrojovém kódu třídy „form-group“. Opět se jedná o styly z Bootstrapu, tentokrát jde o jakýsi kontejner, který obaluje políčka formuláře. Jak je vidět,

následně obsahuje Latte makro „input“, pomocí kterého Nette vygeneruje tag <input> s nastavenými atributy. Z řádku 9 tedy vyrenderuje tag <input type=“email“ class=“form-control“ placeholder=“jan@novak.cz“ required>. Nastavená bootstrapová třída „form-control“ nastaví styl políčka ve formuláři.

Následně je dobré se zaměřit na část s tlačítky pro přihlášení a registraci. Obě tlačítka jsou obaleny v divu s třídami. Jako první je uvedena „btn-group“, která nastavuje styl pro skupinu tlačítek. Následuje třída „btn-group-justified“, která definuje jejich zarovnání a jako poslední je uvedena „logRegBtns“, což už není třída Bootstrapu, ale ručně přidaná do stylů aplikace a upravuje nevhodné nastavení z Bootstrapu. V tomto kontejneru jsou dále tlačítka s třídou „btn“, která nastavuje styl pro tlačítko a „btn-primary“, které dodá obecnému tlačítku další styly. Tato modifikační třída opět není jediná (podobně jako v případě třídy „alert-nějaká_modifikace“). V dokumentaci je možné najít kompletní seznam i s ukázkou.

Na konci této stránky dochází k připojení JavaScriptu. V tomto případě se jedná o spojené a minifikované skripty jQuery a Bootstrapu. Bootstrap zajišťuje například rozbalovací responzivní menu a jQuery je jeho závislostí, která je potřebná pro jeho chod. Tento spojený a minifikovaný soubor byl vytvořen pomocí Gulpu, který bude vysvětlen v dalších částech.

Jak je vidět, pomocí Bootstrapu je možné vytvořit velice snadno hezké uživatelské rozhraní. Nevýhodou může být to, že trvá docela dlouhou dobu, než si člověk na framework zvykne a všechny třídy si zapamatuje. Ze začátku totiž stráví značnou část vývoje hledáním všeho potřebného v dokumentaci.

Zdrojový kód 4.3: Šablona přihlašovacího formuláře

```
1 {layout '../@layout2.latte'}
2 {block content}
3 <h2>Přihlásit se</h2>
4 {foreach $flashes as $flash}
5     <div class="alert alert-{$flash->type}">{$flash->
          message}</div>
6 {/foreach}
7 {form loginForm}
8 <div class="form-group">
```

```
 9   {input email class=>"form-control"}
10 </div>
11 <div class="form-group">
12   {input password class=>"form-control"}
13 </div>
14 <div class="btn-group btn-group-justified logRegBtns">
15   <div class="btn-group">
16     {input send class=>"btn btn-primary"}
17   </div>
18   <div class="btn-group">
19     <a class="btn btn-default" n:href="Sign:registration">
          Registrovat</a>
20   </div>
21 </div>
22 {/form}
23 <script src="/js/bootstrap_jquery.min.js"></script>
```

Hlavní šablona pro registraci a přihlášení

Hlavní šablona pro přihlášení a registraci je umístěna v souboru „@layout2.latte“. Tato šablona obsahuje celou základní strukturu HTML dokumentu (doctype, tagy html, head, body atd.). Zde, ve zdrojovém kódu 4.4 je ukázáno pouze to hlavní a tím je mřížka z Bootstrapu. Jedná se velice užitečnou komponentu pro tvorbu responzivních webů. Tuto mřížku si jde představit jako stránku, která je rozdělena na 12 sloupců a pomocí tříd dochází k nastavení, kolik dvanáctin potřebný element zabírá. V tomto kódu je vidět div, který má nastavenou třídu „container-vertical“, která nastavuje zarovnání obsahu na střed, následuje třída „container“, která je součásti Bootstrapu je dobré ji používat jako ohraničení stránky. Následuje třída „row“ s modifikací „row-login“, která odstraňuje pro tento příklad nevhodné nastavení.

Hlavní částí je , který obsahuje několik tříd s názvem „col-velikost-číslo“. Pro pochopení je potřeba vysvětlit následující. Bootstrap má nastaveny 4 záchytné body pro šířku displaye a ty jsou:

- xs – extra small

- sm – small
- md – medium
- lg – large

Použité třídy tedy nastavují, kolik dvanáctin z šířky dané velikosti displaye bude sloupec zabírat. Tyto šířky jde v nastavení Bootstrapu nastavit podle potřeby, v tomto příkladu jsou ale použity standardní. Další třídou je „col-velikost-offset-číslo“, která značí o kolik dvanáctin se sloupec posune doprava.

Zdrojový kód 4.4: Mřížka v šabloně @layout2.latte

```
<div class="container-vertical">
  <div class="container">
    <div class="row row-login">
      <div class="col-xs-12 col-sm-8 col-sm-offset-2 col-
          md-6 col-md-offset-3 col-lg-6 text-center">
        {include content}
      </div>
    </div>
  </div>
</div>
```

V tomto příkladu je tedy nastaveno, že hlavní element, obalující vložený obsah (registrační a přihlašovací formulář) bude zabírat na velmi malém displayi 12 dvanáctin šířky (tedy celou šířku), na malém 8 dvanáctin, na středním 6 dvanáctin a velkém 6 dvanáctin. Tento element je následně pomocí offsetu posunut doprava o potřebný počet dvanáctin šířky, aby se zobrazoval na prostředku.

Zdrojový kód 4.5: Vertikální zarovnání kontejneru

```
.container-vertical{
  height: 100vh;
  display: flexbox;
  display: flex;
  width: 100vw;
  align-items: center;
  min-height: 400px;
}
```

U této šablony je jako poslední dobré ukázat vertikální zarovnání kontejneru. CSS styly jsou ve zdrojovém kódu 4.5. Zde je vidět využití novinky v CSS – flexboxu. Vertikální zarovnání býval v minulosti problém, který se řešil různými hacky, ale zde je krásně vidět, jak se CSS vyvíjí k lepšímu. Stačí hlavnímu divu nastavit šířku a výšku pomocí nových jednotek vw a vh, nastavit „display“ na „flex“ a „align-items“ na „center“. Dále je dobré si všimnout fallbacku pro IE 10, které „display: flex“ nepodporuje, ale podporuje „display:flexbox“. Novější prohlížeče vezmou to nastavení, které je uvedeno jako poslední, zatímco starší prohlížeče to novější neumí a proto ho ignorují.

Registrace

Registrační stránka se do značné míry podobá přihlašovací stránce a proto zde nebude tak detailně rozebírána jako stránka předchozí, ale lze jí nalézt na přiloženém CD.

Zdrojový kód 4.6: Továrnička registračního formuláře

```
1  public function createComponentRegistrationForm(){
2    $form=new Form;
3    $form->addText('email')
4          ->addRule(Form::EMAIL, 'E-mailová adresa není
               platná')
5          ...
6    $form->addPassword('password')
7          ->addRule(FORM::MIN_LENGTH, 'Heslo musí obsahovat
               alespoň 6 znaků', 6)
8          ...
9    $form->addPassword('password2')
10         ->addRule(FORM::EQUAL, 'Zadaná hesla se musí
               shodovat', $form['password'])
11         ...
12 }
```

Za zmínku ale stojí její továrnička (zdrojový kód 4.6) v presenteru, kde je vidět, jak lze pomocí Nette frameworku ošetřovat zadané hodnoty. Zde je vidět, že Nette

má již určitá pravidla přednastavená (minimální délka, zadaný typ, rovnost atd.), ale je možné využívat i regulární výrazy.

Hlavní strana

Nyní přichází na řadu hlavní strana, tedy stránka, která obsahuje správu osobních výdajů. Ta je narozdíl od registrační a přihlašovací stránky zasazena do defaultní šablony „@layout.latte“ a je zobrazena ve zdrojovém kódu 4.7.

Zdrojový kód 4.7: Šablona hlavní strany

```
1  {block content}
2  <div class="container">
3    {include '../nav.latte'}
4    <h2>Přehled</h2>
5    <div id="tableDashboardComponent"></div>
6  </div>
7  <script src="/js/bootstrap_jquery.min.js"></script>
8  <script src="/js/react.min.js"></script>
9  <script src="/js/Chart.min.js"></script>
10 <script src="/js/dashboardTable.min.js"></script>
```

Tato ukázka vypadá na první pohled velice jednoduše. Je zde umístěn obalovací element s třídou „container“, do které je naimportována šablona „nav.latte“ a následují nadpis druhé úrovně a s „id“ nastaveném na „tableDashboardComponent“. To celé ja zkončeno připojením javascriptových souborů pro Bootstrap a jQuery, React, javascriptový plugin Chart.js a další javascriptový soubor, speciální pro tuto stránku, který obsahuje komponentu, fungující díky knihovně React.

V této části bude zaměřena pozornost právě na výše zmiňovaný soubor „dashboardTable.min.js“, přesněji řečeno na jeho nezkompilovanou a neminifikovanou verzi. Tento soubor obsahuje nastavení 2 „reactích“ komponent – přehledové tabulky a editačního formuláře.

Zaměření bude nejprve na přehledovou tabulku. Kostra je vidět na zdrojovém kódu 4.8, kde je možné vidět definice jednotlivých funkcí. Celý zdrojový kód je příliš velký a proto bude vysvětlován po částech a jako celek je umístěn v příloze A.5.

Popis jednotlivých funkcí je následující:

- getInitialState – nastavení počátečních stavů komponenty
- componentDidMount – činnosti, které se mají vykonat po načtení komponenty
- handleChangeMonth – změna stavu komponenty po změně měsíce
- handleChangeYear – změna stavu komponenty po změně roku
- handleChangePage – změna stránky (pokud existuje tolik položek, že je zaplé stránkování)
- handleAddForm – zobrazení přidávacího formuláře
- handleAddFormRem – skrytí přidávacího formuláře
- handleEditForm – zobrazení editačního formuláře
- handleEditFormRem – skrytí editačního formuláře
- handleClickDelete – zobrazení dotazu, jestli chce uživatel záznam opravdu smazat
- formatDate – převede datum ve tvaru YYYY-MM-DD na DD. MM. YYYY
- render – povinná funkce, která renderuje komponentu

Zdrojový kód 4.8: Kostra komponenty přehledové tabulky

```
var TableDashboardComponent = React.createClass({
  getInitialState: function() {...},
  componentDidMount: function() {...},
  handleChangeMonth: function(event) {...},
  handleChangeYear: function(event) {...},
  handleClickChangePage: function(page){...},
  handleAddForm: function() {...},
  handleAddFormRem: function() {...},
  handleEditForm: function(item) {...},
  handleEditFormRem: function() {...},
  handleClickDelete: function(item) {...},
  formatDate: function(date){...},
  render: function() {...}
});
```

Jelikož je již kostra známá a jednotlivé funkce jsou alespoň základně popsány, může dojít k detailnějšímu rozboru těch částí, které jsou potřeba. Metoda „getInitialState“

pouze nastavuje počáteční stavy aplikace a pohledem do zdrojového kódu by měla být dostatečně srozumitelná. O něco horší to je s metodou „componentDidMount“, která načítá potřebná data a nastavuje stavy. Její obsah je vidět ve zdrojovém kódu 4.9.

Zdrojový kód 4.9: Obsah metody componentDidMount

```
1 $.ajax({
2   type:'POST',
3   url: "?do=getCosts",
4   dataType: "json",
5   data: {...},
6   success: function(data) {
7     this.setState({...});
8   }.bind(this)
9 });
```

Tato metoda tedy pouze odesílá pomocí jQuery AJAXový požadavek na adresu „?do=getCosts“, jako parametr odesílá měsíc, rok a stránku (ze stránkování, pokud existuje), očekává odpověď v JSON formátu a pokud vše proběhne jak má, nastaví stavy komponenty na obdržená data.

Další metody, které se do značné míry podobají metodě „componentDidMount“ jsou „handleChangeMonth“, „handleChangeYear“ a „handleClickChangePage“, protože jejich hlavní funkcí je také odesílat AJAXový požadavek na určité adresy s určitými parametry. Jako odpověď také obdrží JSON a podle výsledku obnoví stavy komponenty. Hlavní změna je v tom, že ještě před AJAXovým požadavkem dochází ke změně stavu podle nově vybrané hodnoty. Zkrácená ukázka je vidět ve zdrojovém kódu 4.10

Zdrojový kód 4.10: Změna měsíce, roku a stránky

```
1 handleChangeMonth: function(event) {
2   this.setState({ month: event.target.value });
3   $.ajax({...});
4 },
5 handleChangeYear: function(event) {
```

```
6     this.setState({ year: event.target.value });
7     $.ajax({...});
8   },
9   handleClickChangePage: function(page){
10    $.ajax({...});
11  }
```

Další metody nejsou nijak složité a proto zde je jen malá ukázka zdrojového kódu (v případě potřeby je možné najít celý v příloze A.5).

- handleAddForm – tato metoda pouze mění stav komponenty „showAddForm“ na true a tím v renderu dojde k vykreslení přidávacího formuláře
- handleAddFormRem – jedná se o metodu, která je volána z komponenty „EditForm“, použité pro přidání a mění stav komponenty „showAddForm“ na false a tím zavírá přidávací formulář
- handleEditForm – tato metoda nastavuje do stavu „editForm“ celou komponentu editovacího formuláře, které předává jako potřebná data jako parametr a v editovacím formuláři budou vidět předvyplněné hodnoty, které se mají měnit, ukázka je ve zdrojovém kódu 4.11
- handleEditFormRem – tato metoda je volána z editovací komponenty, kterou po zavolání zavírá
- handleClickDelete – tato metoda je volána ve chvíli, kdy je kliknuto na mazací tlačítko, tím se vyvolá potvrzovací formulář, po jehož potvrzení dojde k odeslání AJAXového požadavku, který zajistí smazání a vrátí novou sadu dat, která je posléze nastavená do stavu komponenty

Zdrojový kód 4.11: Nastavení komponenty editovacího formuláře

```
1  handleEditForm: function(item) {
2    this.setState({editForm: <EditForm data={item}
       clickHandler={this.handleEditFormRem} type="edit"
       />});
3    this.setState({showEditForm: true});
4  }
```

V tuto chvíli je možné přejít na poslední metodu, kterou je „render“. V této

metodě je možné zpracovat několik posledních věcí před tím, než bude komponenta vyrenderována. Tato metoda nakonec vrací výslednou podobu komponenty. Ukázka je vidět ve zdrojovém kódu 4.12. Obsah metody má opět určité, méně důležité části vynechané a celý lze nalézt v přílohách. Úplně na začátku dochází k nastavení proměnných, se kterými se bude pracovat (vynecháno). Poté se zjistí podle stavu, jestli má dojít k zobrazení přidávacího formuláře a pokud ano, uloží se do proměnné. To samé se děje hned poté u editovacího formuláře, akorát v tomto případě je komponenta uložena ve stavu. Následuje rozhodnutí, zda zobrazit stránkování (tato část bude detailněji rozebrána dále). Poté dochází k nastavení grafu (také bude vysvětleno dále) a nakonec, těsně před vrácením výsledné hodnoty se ověřuje, zda nějaké záznamy existují a jestli se nemá zobrazit hláška o prázdné tabulce.

Zdrojový kód 4.12: Renderovací metoda komponenty TableDashboardComponent

```
1  render: function() {
2    ...
3    if (this.state.showAddForm==true){
4      addForm = <EditForm clickHandler={
           this.handleAddFormRem} type="add" month={
           this.state.month} year={this.state.year} page={
           this.state.page} />;
5    }
6    if (this.state.showEditForm==true){
7      editForm = this.state.editForm;
8    }
9    if (this.state.data!=null) {
10     if (this.state.pages>1) {...}
11   }
12   ...
13   var empty=null;
14   if (this.state.data.length==0) empty = <tr><td colspan
       ="6">Nejsou žádné záznamy</td></tr>;
15   return ...;
16 }
```

Nyní je potřeba popsat způsob, jakým se vypisuje stránkování. Ukázka je ve

zdrojovém kódu 4.13. Nejprve dojde k vytvoření pole, do kterého se posléze uloží hodnoty jednotlivých stránek, přičemž se vychází z hodnoty, která je uložena ve stavu komponenty. Když je pole úspěšně vytvořeno, dochází k tvorbě samotného stránkování. Díky Reactu je možné provést zápis, jaký je uveden v ukázce. Do proměnné pagination se uloží seznam, ve kterém se pomocí metody „map“ (obdoba foreach) prochází všechny hodnoty v poli a tvoří se jednotlivé položky seznamu, kterým se přiřazují třídy. Během toho dochází ke kontrole, zda zrovna tato stránka není ta co je aktivní (podle uloženého stavu). Pokud je, nastaví se třída „active“. Samotné položce seznamu se také přiřadí, jaká metoda se volá po kliknutí na změnu stránky.

Zdrojový kód 4.13: Nastavení stránkování

```
1  var pagesArray = new Array();
2  for(var i=1;i<=this.state.pages;i++)
3  {
4    pagesArray.push(i);
5  }
6  pagination = <ul className="pagination pull-center">
7    {
8      pagesArray.map(function(page) {
9        return <li key={page} className={(this.state.page ==
             page ? 'paginationBtn active' : 'paginationBtn')
           }><a onClick={this.handleClickChangePage.bind(
           this,page)}>{page}</a></li>
10     }, this)
11   }
12 </ul>;
```

Ve zdrojovém kódu 4.13 si je také možné všimnout tříd „pagination“ a „pull-center“. Jsou to opět třídy, pocházející z Bootstrapu, kde první zmíněná nastaví seznamu styly pro stránkování a druhá ho posune na střed.

Ve zdrojovém kódu 4.14 je znázorněno vykreslení grafu. Nejprve dojde k vytvoření kontejneru, který se vloží do stránky. Následně se připraví objekt „data“, který obsahuje potřebná nastavení grafu (pro vykreslení grafu je použita knihovna

Chart.js). Následně dochází k vytvoření nové instance grafu, která je připojena do předem vytvořeného kontejneru.

Zdrojový kód 4.14: Vykreslení grafu

```
1  $('.chartCanvasContainer').append('<canvas id="
       categoryChartCanvas" width="300" height="300"></canvas
       >');
2  var ctx = document.getElementById('categoryChartCanvas');
3  if (ctx!=null && this.state.categoriesData!=null) {
4    var data = {
5        labels:  this.state.categoriesLabels,
6        datasets: [
7          {
8            data: this.state.categoriesData,
9            backgroundColor: colorSet,
10         }]
11   };
12   var categoryChart = new Chart(ctx, {
13     data: data,
14     ...
15   });
16 }
```

Kromě návratové hodnoty metody render bylo již zmíněno vše potřebné. Jelikož tato část kódu je velice dlouhá, je možné ji nalézt v příloze A.5. Dochází ke vrácení HTML kódu, obohaceného o různé proměnné. Jelikož princip by měl být jasný z předchozích částí, nebude se tato část návratovou hodnotou detailně zabývat. Jediné, co by bylo dobré zmínit jsou opět použité třídy z Bootstrapu, konkrétně „panel“, „panel-default“, „table“ a „table-condensed“. První a druhá zmíněná tvoří ohraničení okolo tabulky a třetí a čtvrtá nastavuje styly, které jsou u tabulky použity.

Další komponentou, která je na této stránce použita je AddForm. Značná část je podobná již dříve rozebírané komponentě a proto zde bude uvedeno jen to nové. Celý zdrojový kód je možné nalézt v příloze A.5. Tato komponenta obsahuje následující

metody:

- getInitialState
- handleChangeDate
- handleChangeDescription
- handleChangePrice
- handleChangeCategory
- componentDidMount
- handleSave
- render

Jak je vidět už podle názvů, v principu je velká podobnost s předchozí komponentou, ale tato je podstatně jednodušší. Za zmínku stojí to, že tato komponenta se zavolá z komponenty TableDashboardComponent a zobrazuje formulář pro přidání nebo editaci záznamu. Proto se jako parametr předává typ a v určitých metodách jsou uvedeny podmínky.

Když byla na začátku řeč o šabloně „@layout.latte“, bylo zmíněno, že v ní dochází k importu souboru „nav.latte“. Tento soubor obsahuje hlavičku stránky a všechna menu viz zdrojový kód 4.15. Zde je vidět, že v tagu <nav> se třídami z Bootstrapu jsou umístěny 2 tagy . Tato část využívá komponentu Bootstrapu, která se stará o vykreslování responzivních navigací. Jak je vidět, třídy jsou pojmenovávány stále podobným stylem a není problém odhadnout, co jaká třída nastavuje.

Zdrojový kód 4.15: Responzivní menu

```
1 <nav class="navbar navbar-default">
2   <div class="navbar-header">
3     <button type="button" class="navbar-toggle collapsed
        navbar-btn" data-toggle="collapse" data-target="#bs
        -navbar" aria-expanded="false">
4       <span class="icon-bar"></span>
5       <span class="icon-bar"></span>
6       <span class="icon-bar"></span>
7     </button>
8     <h1 class="h1-loged">MoneyTrack</h1>
9   </div>
```

```
10    <div class="collapse navbar-collapse" id="bs-navbar">
11      <ul class="nav navbar-nav navbar-right">
12        <li><a n:href="About:default">O aplikaci</a></li>
13        <li><a n:href="Contact:default">Kontakt</a></li>
14        <li><a n:href="Accounts:default">Můj účet</a></li>
15        <li><a n:href="Sign:out">Odhlásit se</a></li>
16      </ul>
17    </div>
18  </nav>
```

Kromě výše zmíněného menu zde existuje ještě jedno, kde se přepíná mezi přehledem a kategoriemi. Vše je vidět ve zdrojovém kódu 4.16. Jak je vidět, toto menu opět obsahuje třídu „nav“, ke které je ale přidána „nav-tabs“. Dojde tím tedy k nastavení typu na záložkové.

Zdrojový kód 4.16: Záložkové menu

```
1  <ul class="nav nav-tabs costRepBtns">
2    <li {ifset $costs}class="active"{/ifset}><a n:href="
       Dashboard:default">Náklady</a></li>
3    <li {ifset $categories}class="active"{/ifset}><a n:href
       ="Categories:default">Kategorie</a></li>
4    </ul>
```

O aplikaci

Stránka obsahující informace o aplikaci slouží především jako ukázka nového atributu „srcset“ u obrázku. Vše je vidět ve zdrojovém kódu 4.17. Na této stránce je v textu zobrazen obrázek, který načítá zdroj podle toho, jak je široký display. Když je šířka displaye do 768px, načte se „aboutImage@250.jpg“, pokud bude šířka do 992px, načte se „aboutImage@350.jpg“ a takto to pokračuje. Pro prohlížeče, které srcset nepodporují je zde fallback v podobě klasického „src“ atributu.

Zdrojový kód 4.17: O aplikaci

```
1  <div class="container">
2    ...
```

```
3   <p class="pAbout">
4     <img src="/images/aboutImage@500.jpg" srcset="/images/
        aboutImage@250.jpg 768w, /images/aboutImage@350.jpg
         992w, /images/aboutImage@500.jpg 1200w" alt="
        MoneyTrack" width="40%" class="aboutImage hidden-xs
        ">
5     {$about|breaklines}
6   </p>
7   <div class="clearfix"></div>
8 </div>
```

Kontakt

Stránka „Kontakt“, obsahuje 2 bloky, které se zobrazují podle toho, na jakém zařízení je stránka zobrazena. První blok obsahuje kontaktní informace a druhý kontaktní formulář. Specialitou je to, že kromě mobilů se má zobrazovat první kontaktní formulář. V tomto případě se dá opět využít síla Bootstrapu, jak je vidět na zdrojovém kódu 4.18.

Zdrojový kód 4.18: Kontakt

```
1 <div class="container">
2   ...
3   <div class="row">
4     <div class="col-xs-12 col-sm-6 col-sm-push-6
        contactsBlock">
5       {$contact|noescape}
6     </div>
7     <div class="col-xs-12 col-sm-6 col-sm-pull-6
        contactsFormBlock">
8       ...
9     </div>
10  </div>
11 </div>
```

Ostatní stránky

Výše byl podrobně rozebrán zdrojový kód několika stránek. Zdrojové kódy dalších stránek jsou umístěny v přílohách a na CD, protože vše potřebné pro jejich fungování již bylo vysvětleno na těch předchozích. Stránka s kategoriemi je v principu velice podobná hlavní straně s přehledem. Také obsahuje tabulku, do které lze přidávat položky a také využívá knihovnu React a AJAXové požadavky. Stránka „Můj účet“ pro změnu obsahuje formulář, který zajišťuje změnu hesla.

LESS a CSS

Pro kódování kaskádových stylů byl v této aplikaci použit preprocesor LESS, který byl následně kompilován do CSS. Hlavním souborem je „style.less“, jehož zkrácená verze je vidět ve zdrojovém kódu 4.19.

Zdrojový kód 4.19: style.less

```
1  @import "variables\_overload.less";
2  ...
3  @media (min-width:@screen-sm) {
4    @import "style-sm.less";
5  }
6  @media (min-width:@screen-md) {
7    @import "style-md.less";
8  }
9  @media (min-width:@screen-lg) {
10   @import "style-lg.less";
11 }
```

Do tohoto souboru je hned na začátku naimportován soubor „variables_overload.less“, který je uveden ve zdrojovém kódu 4.20 a obsahuje nastavení proměnných, se kterými se bude pracovat. Jejich využití je hned následné. V souboru „style.less“ je krásně vidět mobile-first přístup. Na konci souboru jsou totiž postupně importovány další LESS soubory, které jsou obaleny do media queries. Hlavní styly jsou tedy obsaženy ve „style.less“ a ty jsou v případě, že uživatel přistupuje ze zařízení s větším displayem přepisovány. Kromě toho je zde vidět i užitečnost proměnných, které běžné

CSS nepodporuje.

Zdrojový kód 4.20: variables_overload.less

```
@screen-sm: 768px;
@screen-md: 992px;
@screen-lg: 1200px;
@icon-font-path: "fonts/";
```

Kromě výše zmíněných LESS souborů se využívají i další a přímo souvisejí s Bootstrapem. Hlavním tímto souborem je „bootstrap.less“, jehož zkrácený obsah je ve zdrojovém kódu 4.21 (celá verze je v příloze A.2). Jelikož je Bootstrap napsán v LESSu a tato verze je přítupná, je možné si z Bootstrapu vytáhnout jen ty nejdůležitější části a vše nepotřebné ignorovat. V tomto souboru je hned na začátku vidět import souboru „variables_overload.less“, který přepíše proměnné z původního souboru Bootstrapu. Následně dochází k importu jen opravdu potřebných komponent.

Zdrojový kód 4.21: bootstrap.less

```
// Core variables and mixins
@import "../assets/bootstrap/less/variables.less";
@import "variables_overload.less";
@import "../assets/bootstrap/less/mixins.less";
// Reset and dependencies
@import "../assets/bootstrap/less/normalize.less";
...
// Core CSS
@import "../assets/bootstrap/less/scaffolding.less";
...
// Components
@import "../assets/bootstrap/less/component-
   animations.less";
...
// Utility classes
@import "../assets/bootstrap/less/utilities.less";
@import "../assets/bootstrap/less/responsive-
   utilities.less";
```

4.3.6 Práce s Gulpem

V předchozí sekci byl zmíněn preprocesor LESS, který je ale potřeba nějakým způsobem převést do klasického CSS, aby byly styly čitelné pro prohlížeče. Díky ekosystému Node.js a javascriptovým taskrunnerům není problém podobné činnosti zautomatizovat.

Při programování této aplikace byly použity především 2 aplikace z Node.js ekosystému – Bower a Gulp. Bower slouží k instalaci různých frontendových modulů (v tomto případě jQuery, React atd.) a vše potřebné již bylo popsáno v teoretické části. Tato část bude tedy věnovaná Gulpu.

Aby bylo možné Gulp používat, je potřeba přes příkazovou řádku otevřít adresář, ze kterého se bude Gulp spouštět a v tomto adresáři spustit příkaz „npm init“. Po vyplnění průvodce dojde k inicializaci balíčkovacího systému NPM a bude možné Gulp nainstalovat příkazem „npm install gulp –save-dev“. Po úspěšné instalaci stačí stejným způsobem instalovat různé potřebné moduly a vytvořit soubor „gulpfile.js“, do kterého se píše konfigurace.

Kompilace LESS do CSS

První činností, kterou Gulp velice usnadní je kompilace LESSu do CSS. Prtvní co se v konfiguračním souboru musí nastavit, jsou jednotlivé nainstalované moduly do proměnných (ukázka ve zdrojovém kódu 4.22).

Zdrojový kód 4.22: Zavedení modulů Gulpu

```
1 var gulp        = require('gulp');
2 var minify      = require('gulp-minify');
3 var less        = require('gulp-less');
4 var concat      = require('gulp-concat');
5 var cssmin      = require('gulp-cssmin');
6 var autoprefixer = require('gulp-autoprefixer');
7 var responsive = require('gulp-responsive-images');
```

Ve chvíli, kdy jsou jednotlivé moduly připojeny, je možné tvořit úlohy, jejichž konfigurace je vidět ve zdrojovém kódu 4.23. V tomto příkladu jsou vytvořeny 3

úlohy. první, která se jmenuje „less“ převede pomocí modulu „gulp-less“ soubor na CSS, do adresáře „./css/“. Druhá úloha „bs_less2css“ zpracuje stejným způsobem soubor „bootstrap.less“ a třetí tyto soubory vezme, spojí do jednoho souboru, přidá prefixy CSS vlastností pro starší prohlížeče, minifikuje a nakonec výsledný soubor umístí do produkčního adresáře.

Zdrojový kód 4.23: Kompilace LESS do CSS

```
gulp.task('less', function() {
    return gulp.src("less/style.less")
        .pipe(less())
        .pipe(gulp.dest("./css/"));
});
gulp.task('bs_less2css', function() {
    return gulp.src("less/bootstrap.less")
        .pipe(less())
        .pipe(gulp.dest("./css/"));
});
gulp.task('compileCSS', function(){
    return gulp.src("css/*.css")
        .pipe(concat("main.css"))
        .pipe(autoprefixer())
        .pipe(cssmin())
        .pipe(gulp.dest("../prod/"));
});
```

Zpracování JavaScriptu

Kromě kompilace LESSu byl Gulp použit také ke zpracování finálních javascriptových souborů. První použitá úloha je zobrazena ve zdrojovém kódu 4.24, která vezme jako zdroj soubory, obsahující komponenty pro evidenci nákladů a kategorií. Tyto soubory načte, převede z JSX na JS, minifikuje a výsledný soubor umístí do produkční verze.

Zdrojový kód 4.24: Zpracování kódu komponent Reactu

```
gulp.task('compileJS', function(){
```

```
2   return gulp.src(["js/categories.js","js/
        dashboardTable.js"])
3     .pipe(react())
4     .pipe(minify({ext: { min: ".min.js" }}))
5     .pipe(gulp.dest("../prod/js/"))
6 });
```

Další možností, jak zpracovávat JavaScript je úloha uvedená ve zdrojovém kódu 4.25. Tato úloha má na vstupu několik souborů, které sloučí, minifikuje a umístí do produkční verze. Z tohoto příkladu je také patrné, že dochází ke zpracování pouze části bootstrapového JavaScriptu a není tedy nutné používat celý framework.

Zdrojový kód 4.25: Zpracování zbylého JavaScriptu

```
1 gulp.task('compileJS_bs', function(){
2   return gulp.src(["assets/jquery/dist/jquery.min.js","
        assets/bootstrap/js/collapse.js","assets/bootstrap/js
        /transition.js","assets/jquery.placeholder/
        jquery.placeholder.min.js","js/netteForms.min.js","js
        /index.js"])
3     .pipe(concat("bootstrap_jquery.js"))
4     .pipe(minify({ext: { min: ".min.js" }}))
5     .pipe(gulp.dest("../prod/js/"))
6 });
```

Další možnosti

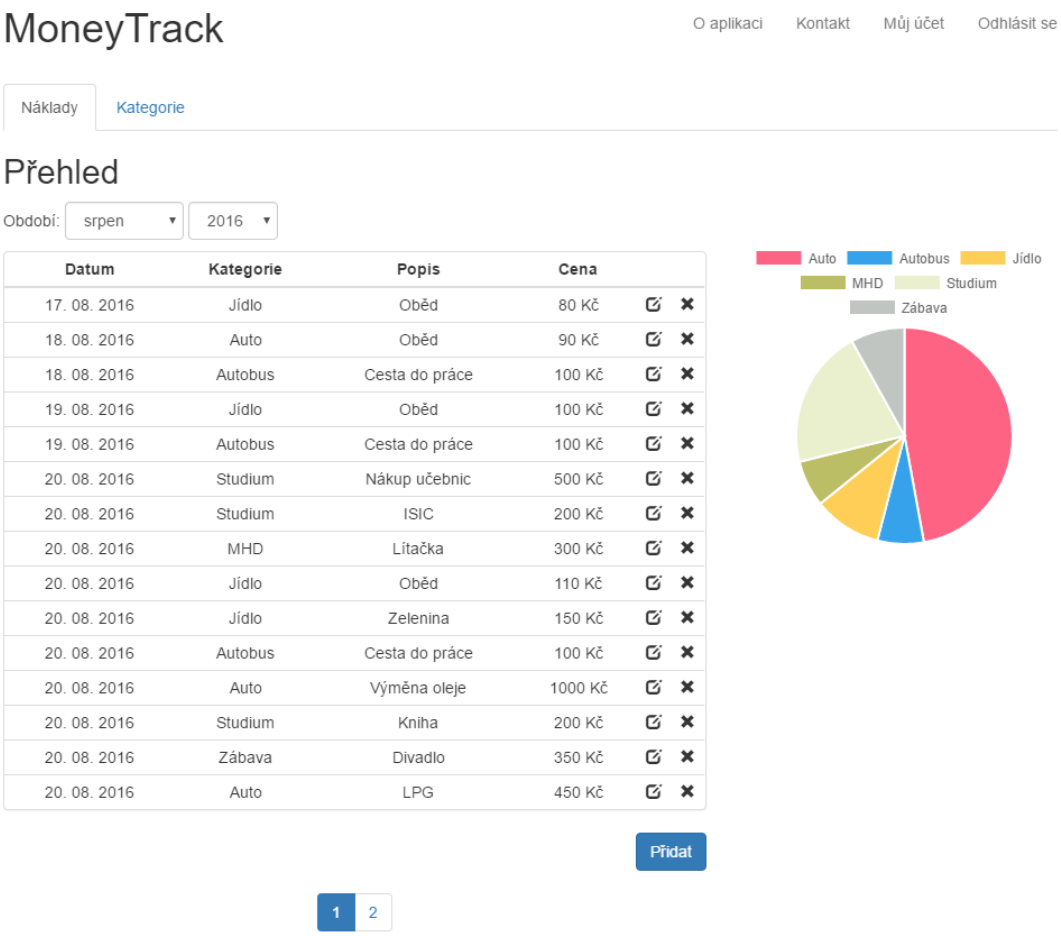
Výše byly zmíněny úlohy, které mají za cíl kompilovat, spojovat a minifikovat JS a CSS soubory. Gulp toho ale samozřejmě umí víc a záleží na uživateli, jak ho využije. Dá se použít například také pro změny velikostí a minifikace obrázků (vývojář nemusí například obrázky zmenšovat ručně a pro minifikaci používat služby jako tinyjpg.com/tinypng.com). Velice užitečná je například funkce „watch“, která sleduje změny v souborech a když změnu zjistí, spustí určitou úlohu (je možné spouštět kompilace LESSu ve chvíli, kdy se soubor změní). Jako poslední plugin, který může většině vývojářů zpříjemnit práci je BrowserSync, což je plugin, který dokáže

po změně v nějakém souboru obnovit automaticky okno prohlížeče a kromě toho vytvoří vlastní webový server a externí webovou adresu, takže vývojář může web testovat z dalších zařízení, která jsou připojena do sítě.

4.4 Nasazení na server a testování

Pro účely této diplomové práce byl založen webový hosting u Forpsi. Jelikož byl backend aplikace naprogramován v PHP, stačilo pouze nakopírovat aplikaci na server pomocí FTP klienta, konkrétně WinSCP, jehož velikou výhodou je, že dokáže hlídat změny souborů a v případě změny je uploadovat.

Obrázek 4.5: Screenshot počítačové verze



Zdroj: vlastní zpracování

Aplikace byla testována na různých fyzických zařízeních, které byly k dispozici a také pomocí trial verze služby BrowserStack, která je popsána v teoretické části. Bohužel, pokud nemá vývojář zaplacenou plnou verzi, nejsou dostupné všechny prohlížeče a zařízení. Na všech testovaných prohlížečích a zařízeních se aplikace chovala správně a dala se bez problémů používat i na pomalém mobilním 2G připojení. Screenshot aplikace spuštěné na počítači je vidět na obrázku 4.5 a screeshot z mobilu na obrázku 4.6.

Obrázek 4.6: Screenshot mobilní verze

Zdroj: vlastní zpracování

5 Závěr

Diplomová práce byla vypracována přesně podle zadání. V teoretické části jsou shrnuty ty nejdůležitější novinky ze světa webového frontendu. Vše se vesměs točí okolo responzivity a optimalizace pro mobilní zařízení. V úvodu do frontendu jsou rozebrány novinky z HTML5, CSS3 a JavaScriptu, na což je následně napojen popis technologií, pomocí kterých je možné psaní tohoto kódu zefektivnit. Řeč je především o CSS preprocesorech jako LESS, SASS a Stylus, na což navazuje popis javascriptových knihoven jQuery, React a Angular. Jelikož tyto preprocesory a knihovny mnohdy rozšiřují základní zápisy JS a CSS, je potřeba vývojové soubory kompilovat, k čemuž se dá velice hezky využít některého „javascriptového task runneru“, jakým je například Gulp nebo Grunt. Samozřejmě nesměla chybět ani zmínka o verzovacím systému GIT a frontendových frameworcích, zejména tomu nejoblíbenějšímu – Bootstrapu. A jelikož značná část obyvatel České republiky nemá zatím na většině území přístup k rychlým mobilním připojením 3G a 4G, nesměly chybět ani tipy pro zvýšení rychlosti načítání a zobrazování webových stránek.

Cílem práce bylo také v praktické části vytvořit webovou aplikaci, která bude responzivní, optimalizována pro mobilní zařízení a bude na ní možné ukázat současné trendy kódování webového frontendu. Pro tento účel byla vybrána aplikace pro evidenci osobních výdajů. Vývoj se držel tím co bylo předem nastudováno a zaznamenáno v teoretické části. Jako jazyky byly použity samozřejmě HTML5, CSS3 a JavaScript, ve spolupráci s preprocesorem LESS, knihovnami jQuery a React a frontendovým frameworkem Bootstrap. Aplikace také využívá technologie AJAX, díky které je možné posílat a obdržovat informace ze serveru, aniž by muselo docházet k obnovení stránky. Ke kompilaci zdrojového kódu byl použit Gulp, díky kterému bylo možné pospojovat a minifikovat jednotlivé soubory, což web značně zrychlilo. V praktické části je podrobně popsán celý postup, včetně zkrácených ukázek zdrojových kódů (kompletní zdrojové kódy lze nalézt v přílohách). Výsledná aplikace byla otestována pomocí služby BrowserStack na různých prohlížečích a také na různých dostupných mobilních zařízeních. Výsledky byly velice dobré, protože aplikaci bylo možné používat na všech testovaných prohlížečích a na pomalém 2G připojení.

Z výše zmíněného je patrné, jak veliký skok se za poslední roky udál v oblasti kódování webových stránek. Díky nutnosti optimalizovat weby pro prakticky neomezené množství velikostí displayů a s tím zvýšené nutnosti automatizace a usnadnění určitých částí vývoje, začaly vznikat další a další nástroje, které se přesně o toto snažily. Bohužel jich je v dnešní době takové množství, že se začínající frontendový vývojář bude zpočátku velice těžce orientovat a dosažení té nejvyšší úrovně ho bude stát podstatně větší čas a úsilí, než tomu bylo před několika lety.

6 Summary

Most people cannot imagine the world without the Internet. It is the biggest computer network, which is great source of information. People read information mostly on web pages, which are stored on web servers and it is accessible from web browser. Every web page has own structure and web development can be divided into two parts – backend and frontend. This thesis is focused on frontend development.

In theoretical part, there is included basic information about news and trends in present frontend web development. Becouse of mobile devices, frontend development is now harder than before and there are many tools which have to be learned for effective development. This information, which is located in the theoretical part is used for practical part, becouse there is described, how could be frontend of web application made. In this part there is described whole process, which begins with a proposal and continues throught development to final testing and deployment and every step is described in detail.

Keywords

Responsive webdesign, Frontend, HTML5, CSS3, JavaScript, React, jQuery, Bootstrap, Node.js, Gulp

Seznam použitých zdrojů

Šťastný, J., & Šimeček, M. (2010). *Css3 - držte krok s dobou (nové vlastnosti).* Dostupné z http://programujte.com/clanek/2010070801-css3-drzte-krok-s-dobou-nove-vlastnosti/

Avocode, Inc. (2016a). *Avocode.* Dostupné z https://avocode.com/

Avocode, Inc. (2016b). *Avocode features.* Dostupné z https://avocode.com/features-designer.html

BrowserStack Limited. (2016). *Browserstack live features.* Dostupné z https://www.browserstack.com/features

Chacon, S. (2009). *Pro git.* Praha: CZ.NIC, z. s. p. o.

Ševčík, M. (2014). *Zurb foundation vs. twitter bootstrap.* Dostupné z http://www.webmotion.cz/blog/webmotion-odborne/zurb-foundation-vs-twitter-bootstrap/

Experti komunity jQuery. (2010). *jquery: Kuchařka programátora.* Brno: Computer Press, a.s.

Krejčí, T. (2015). *Jak zrychlit načítání webu.* Dostupné z https://www.youtube.com/watch?v=xz6ao2pOSi0

Kučera, F. (2011). *Distribuované verzovací systémy – úvod (1).* Dostupné z http://www.abclinuxu.cz/clanky/distribuovane-verzovaci-systemy-uvod-1

Majda, D. (2008). *Do hlubin implementací javascriptu: 1. díl – úvod.* Dostupné z https://www.zdrojak.cz/clanky/do-hlubin-implementaci-javascriptu-1-dil-uvod/

Malý, M. (2010a). *Html5 audio: rádio ve vašich stránkách.* Dostupné z https://www.zdrojak.cz/clanky/html5-audio-radio-ve-vasich-strankach/

Malý, M. (2010b). *Html5: ukládáme si data k elementům.* Dostupné z https://www.zdrojak.cz/clanky/html5-ukladame-si-data-k-elementum/

Michálek, M. (2014a). *Proč adaptivní, ne responzivní. a proč to responzivní zůstane.* Dostupné z http://www.vzhurudolu.cz/blog/23-adaptivni-responzivni

Michálek, M. (2014b). *Průvodce css preprocesory: co a jak?* Dostupné z http://www.vzhurudolu.cz/blog/12-css-preprocesory-1

Michálek, M. (2014c). *Průvodce css preprocesory: který vybrat?* Dostupné z http://

www.vzhurudolu.cz/blog/15-css-preprocesory-4

Michálek, M. (2014d). *Průvodce css preprocesory: technické vlastnosti.* Dostupné z http://www.vzhurudolu.cz/blog/13-css-preprocesory-2

Michálek, M. (2015a). *Critical css.* Dostupné z https://www.youtube.com/watch?v=6x3pSAP-LTo

Michálek, M. (2015b). *Vzhůru do css3.*

Michálek, M. (2016a). *29 užitečných grunt pluginů a zároveň důvodů proč konečně zkusit automatizaci.* Dostupné z http://www.vzhurudolu.cz/prirucka/grunt-pluginy

Michálek, M. (2016b). *Co je to „mobile first"? ale doopravdy.* Dostupné z http://www.vzhurudolu.cz/prirucka/mobile-first

Michálek, M. (2016c). *Grunt.js.* Dostupné z http://www.vzhurudolu.cz/prirucka/grunt

Michálek, M. (2016d). *Instalace node.js ekosystému pro použití na frontendu.* Dostupné z http://www.vzhurudolu.cz/prirucka/node-instalace

Michálek, M. (2016e). *Nástroje co používám pro frontend kodéřinu.* Dostupné z http://www.vzhurudolu.cz/prirucka/nastroje

Mikšů, V. (2016). *React - Úvod.* Dostupné z https://www.dzejes.cz/react-uvod.html

Mozilla Developer Network and individual contributors. (2016a). *Css.* Dostupné z https://developer.mozilla.org/en-US/docs/Web/CSS

Mozilla Developer Network and individual contributors. (2016b). *Introduction to html.* Dostupné z https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Introduction

Mrozek, J. (2012). *Začínáme s angularjs.* Dostupné z https://www.zdrojak.cz/clanky/zaciname-s-angularjs/

Nešetřil, J. (2010). *Javascript na serveru: Začínáme s node.js.* Dostupné z https://www.zdrojak.cz/clanky/javascript-na-serveru-zaciname-s-node-js/

Ožana, R. (2014). *Gulp vs. grunt: souboj bez vítěze a poraženého.* Dostupné z https://www.zdrojak.cz/clanky/gulp-vs-grunt-souboj-bez-viteze-a-porazeneho/

Otto, M. (2016). *Bootstrap.* Dostupné z http://getbootstrap.com/

Refsnes Data. (2016). *Css how to...* Dostupné z http://www.w3schools.com/css/css_howto.asp

Sládek, J. (2010a). *Webdesignérův průvodce po html5 – díl nultý.* Dostupné z https://www.zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-dil-nulty/

Sládek, J. (2010b). *Webdesignérův průvodce po html5 – nová sémantika.* Dostupné z https://www.zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-nova-semantika/

Sládek, J. (2010c). *Webdesignérův průvodce po html5 – nová sémantika ii.* Dostupné z https://www.zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-nova-semantika-ii/

Sládek, J. (2010d). *Webdesignérův průvodce po html5 – pohyblivé obrázky.* Dostupné z https://www.zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-pohyblive-obrazky/

Sládek, J. (2010e). *Webdesignérův průvodce po html5 – taháme data od návštěvníka.* Dostupné z https://www.zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-tahame-data-od-navstevnika/

Sládek, J. (2015). *Na rychlosti záleží?* Dostupné z https://www.youtube.com/watch?v=qsHWH2VLvfw

Steigerwald, D. (2014). *Proč facebook react zabil jquery.* Dostupné z https://www.zdrojak.cz/clanky/proc-facebook-react-zabil-jquery/

Steigerwald, D. (2015). *Proč se vyhnout angularu.* Dostupné z https://www.zdrojak.cz/clanky/proc-se-vyhnout-angularu/

The core less team. (2016). *About.* Dostupné z http://lesscss.org/about/

The Yeoman Team. (2016). *Yeoman.* Dostupné z http://yeoman.io/

Fichtner, Š. (n.d.). *Frontendové frameworky.* Dostupné z http://note.stepanfichtner.cz/frontendove-frameworky/

ZURB, Inc. (2016). *Foundation.* Dostupné z http://foundation.zurb.com/

Seznam obrázků

Seznam zdrojových kódů

Přílohy

A Zdrojové kódy

A.1 LESS soubory

Zdrojový kód A.1: style.less

```
@import "variables_overload.less";

body{
  min-width: 300px;
}
body>.container{
  padding-bottom: 50px;
}
//prihlasovaci formular
.container-vertical{
  height: 100vh;
  display: flexbox;
  display: flex;
  width: 100vw;
  align-items: center;
  min-height: 400px;
}
.row-login{
  margin-top: 100px;
  margin-bottom: 100px;
}
.login-h1{
  margin-top: 0;
  position: absolute;
  text-align: center;
```

```
26    display: block;
27    width: 100%;
28    padding: 20px 0 0 0;
29  }
30  .loginForm-errorList{
31    margin: 0;
32    padding: 0;
33
34    li{
35      list-style-type: none;
36      margin: 0;
37      padding: 0;
38    }
39  }
40  .logRegBtns{
41    margin-bottom: 10px;
42  }
43  //po přihlášení
44  .navbar-default{
45    background: none;
46    border: 0;
47  }
48  .navbar-nav{
49    margin-top: 10px;
50  }
51  .navbar-btn{
52    margin-top: 20px;
53    margin-right: 0;
54  }
55  .navbar-collapse{
56    margin-right: -15px;
57  }
58  .costRepBtns{
59    margin-bottom: 10px;
```

```
}
.form-control-inline{
  display: inline-block;
  width: auto;
}
.panel-default{margin-top: 10px;}
.dashboardTable{
  td, th{
    text-align: center;
  }
  td:nth-child(5),td:nth-child(6){
    width: 30px;
  }
}
//přehled
.dashboardTableContainer{
  padding: 0;
  position: static
}
.deleteBtn{cursor:pointer}
.editBtn{cursor:pointer}
.paginationBtn{cursor:pointer}
.chartCanvasContainer{
  padding: 0 0 0 40px;
}
#categoryChartCanvas{
  padding: 0;
  margin: 0
}
//kategorie
.categoriesContainer{
  padding: 0;
}
.categoriesTable{
```

```
94    td:nth-child(3),td:nth-child(4){
95      width: 30px;
96    }
97  }
98  //editační formulář
99  .editFormContainer{
100   position: absolute;
101   min-width: 100%;
102   min-height: 100%;
103   width: 100vw;
104   height: 100vh;
105   top: 0;
106   left: 0;
107   z-index: 9999;
108   display: block;
109   display: flexbox;
110   display: flex;
111   align-items: center;
112 }
113
114 .editFormContainerIn{
115   position: fixed;
116   min-width: 100%;
117   min-height: 100%;
118   width: 100vw;
119   height: 100vh;
120   top: 0;
121   left: 0;
122   z-index: 99999;
123   background: rgba(0,0,0,.80);
124   display: block;
125   display: flexbox;
126   display: flex;
127   align-items: center;
```

```
}

.editForm{
  padding: 20px;
  background: #fff;
  border-radius: 5px;
  margin-top: 50px;
  margin-bottom: 50px;
}
//můj účet
.pLogged{
  margin: 25px 0;
}
.changePasswordForm{
  padding: 0;
}
.aboutImage{
  float: right;
  margin: 0 0 20px 20px;
}
.pAbout{
  float: left;
}
//kontakt
.contactsBlock table td{
  padding: 0 5px 5px 0;
}
.contactsFormBlock{
  padding-top: 30px;
}
.contactForm-message{
  height: 100px !important;
  resize: none;
}
```

```
@media (min-width:@screen-sm) {
  @import "style-sm.less";
}
@media (min-width:@screen-md) {
  @import "style-md.less";
}
@media (min-width:@screen-lg) {
  @import "style-lg.less";
}
```

Zdrojový kód A.2: bootstrap.less

```
// Core variables and mixins
@import "../assets/bootstrap/less/variables.less";
@import "variables_overload.less";
@import "../assets/bootstrap/less/mixins.less";

// Reset and dependencies
@import "../assets/bootstrap/less/normalize.less";
@import "../assets/bootstrap/less/print.less";
@import "../assets/bootstrap/less/glyphicons.less";

// Core CSS
@import "../assets/bootstrap/less/scaffolding.less";
@import "../assets/bootstrap/less/type.less";
//@import "../assets/bootstrap/less/code.less";
@import "../assets/bootstrap/less/grid.less";
@import "../assets/bootstrap/less/tables.less";
@import "../assets/bootstrap/less/forms.less";
@import "../assets/bootstrap/less/buttons.less";

// Components
@import "../assets/bootstrap/less/component-
   animations.less";
@import "../assets/bootstrap/less/dropdowns.less";
```

```
23 @import "../assets/bootstrap/less/button-groups.less";
24 //@import "../assets/bootstrap/less/input-groups.less";
25 @import "../assets/bootstrap/less/navs.less";
26 @import "../assets/bootstrap/less/navbar.less";
27 //@import "../assets/bootstrap/less/breadcrumbs.less";
28 @import "../assets/bootstrap/less/pagination.less";
29 //@import "../assets/bootstrap/less/pager.less";
30 @import "../assets/bootstrap/less/labels.less";
31 //@import "../assets/bootstrap/less/badges.less";
32 //@import "../assets/bootstrap/less/jumbotron.less";
33 //@import "../assets/bootstrap/less/thumbnails.less";
34 @import "../assets/bootstrap/less/alerts.less";
35 //@import "../assets/bootstrap/less/progress-bars.less";
36 //@import "../assets/bootstrap/less/media.less";
37 @import "../assets/bootstrap/less/list-group.less";
38 @import "../assets/bootstrap/less/panels.less";
39 //@import "../assets/bootstrap/less/responsive-embed.less
    ";
40 //@import "../assets/bootstrap/less/wells.less";
41 //@import "../assets/bootstrap/less/close.less";
42
43 // Components w/ JavaScript
44 //@import "../assets/bootstrap/less/modals.less";
45 //@import "../assets/bootstrap/less/tooltip.less";
46 //@import "../assets/bootstrap/less/popovers.less";
47 //@import "../assets/bootstrap/less/carousel.less";
48
49 // Utility classes
50 @import "../assets/bootstrap/less/utilities.less";
51 @import "../assets/bootstrap/less/responsive-
    utilities.less";
```

Zdrojový kód A.3: style-sm.less

```
1 //login
```

```
2 .login-h1{
3   text-align: left;
4   padding-left: 20px;
5 }
6 .contactsFormBlock{
7   padding-top: 0px;
8 }
```

A.2 JS soubory

Zdrojový kód A.4: categories.js

```
1  var AddForm = React.createClass({
2    //nastavení počátečních stavů
3    getInitialState: function() {
4      if (this.props.type=="edit") {
5        return {
6          id: this.props.data[0],
7          name: this.props.data[1],
8          description: this.props.data[2],
9          update: 0
10       };
11     }
12     else return {
13       name: '',
14       description: '',
15       update: 0
16     }
17   },
18   //změna stavů po změně hodnot
19   handleChangeName: function(event) {this.setState({ name:
         event.target.value });},
20   handleChangeDescription: function(event) {this.setState
        ({ description: event.target.value });},
21   //uložení hodnot
```

```
22  handleSave: function(e){
23    e.preventDefault();
24    if (this.props.type=="edit") {
25      if (this.state.name!=''){
26        $.ajax({
27          type:'POST',
28          url: "/kategorie?do=editCategory",
29          dataType: "json",
30          data: { id: this.state.id, name: this.state.name
                , description: this.state.description }
31        });
32        this.props.clickHandler();
33      }
34      else alert('Název není zadaný');
35    }
36    else {
37      if (this.state.name!=''){
38        $.ajax({
39          type:'POST',
40            url: "/kategorie?do=addCategory",
41            dataType: "json",
42            data: { name: this.state.name, description:
                  this.state.description }
43        });
44          this.props.clickHandler();
45      }
46      else alert('Název není zadaný');
47    }
48  },
49  render: function() {
50    var ret = (
51      <div className="editFormContainer">
52        <div className="editFormContainerIn">
53        <form className="editForm col-xs-10 col-xs-offset
```

```
            -1 col-sm-6 col-sm-offset-3 col-md-4 col-md-
            offset-4" onSubmit={this.handleSave}>
54          <div className="form-group editForm-form-group">
55            <label>Název:</label>
56            <input className="form-control" type="text"
                value={this.state.name} onChange={
                this.handleChangeName} />
57          </div>
58          <div className="form-group editForm-form-group">
59            <label>Popis:</label>
60            <input className="form-control" type="text"
                value={this.state.description} onChange={
                this.handleChangeDescription} />
61          </div>
62          <div className="form-group editForm-form-group
              text-center">
63            <input className="btn btn-primary" type="
                submit" value="Uložit" /> 
64            <input className="btn btn-default" type="
                button" value="Storno" onClick={
                this.props.clickHandler} />
65          </div>
66        </form>
67        </div>
68      </div>);
69
70    return ret;
71  }
72 });
73 //Komponenta kategorií
74 var CategoriesComponent = React.createClass({
75   getInitialState: function() {
76     return {
77       categories: new Array(),
```

```
78        showAddForm: false,
79        editForm: null
80      }
81    },
82    componentDidMount: function() {
83      $.ajax({
84        type:'POST',
85        url: "/kategorie?do=getCategories",
86        dataType: "json",
87        success: function(data) {
88          if (data.categories!=undefined) this.setState({
              categories: data.categories });
89        }.bind(this)
90      });
91    },
92    handleAddNew: function() {
93      this.setState({ showAddForm: true });
94    },
95    //stornování editovacího formuláře
96    removeEditForm: function() {
97      var i = this.state.update+1;
98      this.setState({editForm: ''});
99      this.setState({ update: i});
100     this.componentDidMount();
101   },
102   //skrytí přidávacího formuláře
103   handleAddNewRem: function() {
104     var i = this.state.update+1;
105     this.setState({ showAddForm: false });
106     this.setState({ update: i});
107     this.componentDidMount();
108   },
109   //akce po kliknutí na editaci řádku
110   handleClickEdit: function(item) {
```

```
111     this.setState({editForm: <AddForm data={item}
            clickHandler={this.removeEditForm} type="edit" />})
            ;
112   },
113   //akce, po kliknutí na smazání řádku
114   handleClickDelete: function(item) {
115     if (confirm('Opravdu si přejete smazat kategorii '+
            item[1]+'?')){
116       $.ajax({
117         type:'POST',
118         dataType: 'json',
119           url: "/kategorie?do=deleteCategory",
120           data: { id: item[0] },
121           success: function(data){
122             if (data.categories!=undefined) this.setState
                    ({ categories: data.categories });
123             else this.setState({ categories: new Array()
                    });
124           }.bind(this)
125       });
126     }
127   },
128   render: function() {
129     var addForm = '';
130     if (this.state.showAddForm==true){ //pokud je stav
            zobrazení formuláře true
131       addForm = <AddForm clickHandler={
              this.handleAddNewRem} type="addNew" />;
132     }
133     var editForm=this.state.editForm;
134     var empty=null;
135     if (this.state.categories==0) empty = <tr><td colspan
            ="4">Nejsou žádné záznamy</td></tr>;
136       return <div className="categoriesContainer">
```

```
137             {addForm}{editForm}
138             <div className="panel panel-default">
139               <table className="table table-condensed
                      categoriesTable">
140                 <thead>
141                   <tr>
142                     <th>Název</th>
143                     <th>Popis</th>
144                     <th></th>
145                     <th></th>
146                   </tr>
147                 </thead>
148                 <tbody>
149                     {
150                       this.state.categories.map(function(
                             item) {
151                         return (
152                           <tr key={item[0]}>
153                             <td>{item[1]}</td>
154                             <td>{item[2]}</td>
155                             <td><span className="glyphicon
                                    glyphicon-edit editBtn"
                                    onClick={
                                    this.handleClickEdit.bind(
                                    this, item)}></span></td>
156                             <td><span className="glyphicon
                                    glyphicon-remove deleteBtn"
                                    onClick={
                                    this.handleClickDelete.bind(
                                    this,item)}></span></td>
157                           </tr>
158                         )
159                       }, this)
160                     }
```

```
161                     {empty}
162                   </tbody>
163                 </table>
164               </div>
165
166               <button className="btn btn-primary pull-right"
                     onClick={this.handleAddNew}>Přidat</button>
167             </div>;
168
169     }
170 });
171
172 ReactDOM.render(<CategoriesComponent />,
     document.getElementById('categoriesComponent'));
```

A.3 JS soubory

Zdrojový kód A.5: dashboardTable.js

```
1  //Komponenta editačního formuláře
2  var EditForm = React.createClass({
3    //nastavení počátečních stavů
4    getInitialState: function() {
5      if (this.props.type=="edit") {
6        return {
7          date: this.props.data[2],
8          description: this.props.data[4],
9          price: this.props.data[3],
10         category: this.props.data[1],
11         id: this.props.data[0],
12         categories: null
13       };
14     }
15     else return {
16       date: new Date().toISOString().substring(0, 10),
```

```
17        description: '',
18        price: '',
19        category: '',
20        categories: null
21      }
22    },
23    //změna stavů po změně hodnot
24    handleChangeDate: function(event) {this.setState({ date:
          event.target.value });},
25    handleChangeDescription: function(event) {this.setState
         ({ description: event.target.value });},
26    handleChangePrice: function(event) {this.setState({
         price: event.target.value });},
27    handleChangeCategory: function(event) {this.setState({
         category: event.target.value });},
28    //načtení kategorií AJAXem
29    componentDidMount: function() {
30      $.ajax({
31        type:'POST',
32        url: "kategorie?do=getCategories",
33        dataType: "json",
34        success: function(data) {
35          var categories = new Array();
36          for (var i=0;i<(data.categories.length);i++) {
37            categories.push(data.categories[i]);
38          }
39          this.setState({ categories: categories });
40          if (this.props.type!="edit") this.setState({
               category: categories[0][0]});
41        }.bind(this)
42      });
43    },
44    //uložení hodnot
45    handleSave: function(e){
```

```
46      e.preventDefault();
47      if (this.props.type=="edit") {
48        this.props.clickHandler();
49        if (this.state.date!='' && this.state.category!=''
             && this.state.price!=''){
50          $.ajax({
51            type:'POST',
52            url: "/?do=editCost",
53            dataType: "json",
54            data: { id: this.state.id, date: this.state.date
                 , description: this.state.description,
                 category: this.state.category, price:
                 this.state.price },
55          });
56          this.props.clickHandler();
57        }
58        else alert('Vyplňte všechna povinná pole');
59      }
60      else {
61        if (this.state.date!='' && this.state.category!=''
             && this.state.price!=''){
62          $.ajax({
63            type:'POST',
64            url: "/?do=addCost",
65            dataType: "json",
66            data: { date: this.state.date, description:
                 this.state.description, category:
                 this.state.category, price: this.state.price,
                  month: this.props.month, year:
                 this.props.year, page: this.props.page},
67          });
68          this.props.clickHandler();
69        }
70        else alert('Vyplňte všechna povinná pole');
```

```
71      }
72    },
73    render: function() {
74      var i=0;
75      var selectBox = '';
76      if (this.state.categories!=null) {
77        selectBox =
78        <select className="form-control" value={
              this.state.category} onChange={
              this.handleChangeCategory}>
79          {
80            this.state.categories.map(function(cat) {
81              return <option key={cat[0]} value={cat[0]}>{
                    cat[1]}</option>
82            }, this)
83          }
84        </select>
85      }
86
87      if (this.state.categories!=null) return <div className
           ="editFormContainer">
88          <div className="editFormContainerIn">
89          <form className="editForm col-xs-10 col-xs-offset
              -1 col-sm-6 col-sm-offset-3 col-md-4 col-md-
              offset-4" onSubmit={this.handleSave}>
90            <div className="form-group editForm-form-group">
91              <label>Datum:</label>
92              <input className="form-control" type="date"
                    value={this.state.date} onChange={
                    this.handleChangeDate} />
93
94            </div>
95            <div className="form-group editForm-form-group">
96              <label>Kategorie:</label>
```

```
97              {selectBox}
98            </div>
99            <div className="form-group editForm-form-group">
100             <label>Popis:</label>
101             <input className="form-control" type="text"
                    value={this.state.description} onChange={
                    this.handleChangeDescription} />
102           </div>
103           <div className="form-group editForm-form-group">
104             <label>Cena:</label>
105             <input className="form-control" type="number"
                    value={this.state.price} onChange={
                    this.handleChangePrice} />
106           </div>
107           <div className="form-group editForm-form-group
                 text-center">
108             <input className="btn btn-primary" type="
                    submit" value="Uložit" /> 
109             <input className="btn btn-default" type="
                    button" value="Storno" onClick={
                    this.props.clickHandler} />
110           </div>
111         </form>
112         </div>
113       </div>;
114
115
116     else return null;
117   }
118 });
119 //Dashboard table - komplet
120 var TableDashboardComponent = React.createClass({
121   //nastavení počátečních stavů
122   getInitialState: function() {
```

```
123     return {
124       data: new Array(),
125       activePage: 1,
126       pagesArray: null,
127       month: this.props.month,
128       year: this.props.year,
129       showAddForm: false,
130       showEditForm: false,
131       categoriesLabels: null,
132       categoriesData: null,
133       page: 1,
134       pages: 1,
135       editForm: null,
136       categoriesCount: 0,
137       update: 0
138     }
139   },
140   //načtení dat AJAXem
141   componentDidMount: function() {
142     $.ajax({
143       type:'POST',
144       url: "?do=getCosts",
145       dataType: "json",
146       data: {month: this.state.month, year:
            this.state.year, page: this.state.page },
147       success: function(data) {
148         this.setState({
149           data: data.costs,
150           categoriesLabels: data.categoriesLabels,
151           categoriesData: data.categoriesData,
152           pages: data.pages,
153           categoriesCount: data.categoriesCount
154         });
155       }.bind(this)
```

```
156       });
157     },
158     //změna stavů po změně hodnot
159     handleChangeMonth: function(event) {
160       this.setState({ month: event.target.value });
161       $.ajax({
162         type:'POST',
163         url: "?do=getCosts",
164         dataType: "json",
165         data: {month: event.target.value, year:
              this.state.year, page: 1 },
166         success: function(data) {
167           this.setState({
168             data: data.costs,
169             categoriesLabels: data.categoriesLabels,
170             categoriesData: data.categoriesData,
171             pages: data.pages,
172             page: 1
173           });
174           }.bind(this)
175       });
176     },
177     handleChangeYear: function(event) {
178       this.setState({ year: event.target.value });
179       $.ajax({
180         type:'POST',
181         url: "?do=getCosts",
182         dataType: "json",
183         data: {month: this.state.month, year:
              event.target.value, page: 1 },
184         success: function(data) {
185           this.setState({
186             data: data.costs,
187             categoriesLabels: data.categoriesLabels,
```

```
188           categoriesData: data.categoriesData,
189           pages: data.pages,
190           page: 1
191         });
192       }.bind(this)
193     });
194   },
195   //změna stránky
196   handleClickChangePage: function(page){
197     $.ajax({
198       type:'POST',
199       url: "?do=getCosts",
200       dataType: "json",
201       data: {month: this.state.month, year:
              this.state.year, page: page },
202       success: function(data) {
203         this.setState({
204           data: data.costs,
205           page: page
206         });
207       }.bind(this)
208     });
209   },
210   //zobrazení přidávacího formuláře
211   handleAddForm: function() {
212     if (this.state.categoriesCount==0) alert('Nejprve
          přidejte kategorie');
213     else this.setState({ showAddForm: true });
214   },
215   //skrytí přidávacího formuláře
216   handleAddFormRem: function() {
217     var i = this.state.update+1;
218     this.setState({ showAddForm: false });
219     this.setState({ update: i});
```

```
220     this.componentDidMount();
221   },
222   //akce po kliknutí na editaci řádku
223   handleEditForm: function(item) {
224     this.setState({editForm: <EditForm data={item}
          clickHandler={this.handleEditFormRem} type="edit"
          />});
225     this.setState({showEditForm: true});
226   },
227   //stornování editovacího formuláře
228   handleEditFormRem: function() {
229     var i = this.state.update+1;
230     this.setState({showEditForm: false});
231     this.setState({ update: i});
232     this.componentDidMount();
233   },
234   //akce, po kliknutí na smazání řádku
235   handleClickDelete: function(item) {
236     if (confirm('Opravdu si přejete smazat záznam '+item
          [4]+'?')){
237       $.ajax({
238         type:'POST',
239           url: "?do=deleteCost",
240           dataType: "json",
241           data: { id: item[0], month: this.state.month,
              year: this.state.year, page: this.state.page
              },
242           success: function(data) {
243             this.setState({ pages: data.pages, data:
                data.costs, categoriesLabels:
                data.categoriesLabels, categoriesData:
                data.categoriesData  });
244             if (this.state.page>data.pages) this.setState
                ({ page: data.pages })
```

```
245           }.bind(this)
246       });
247     }
248   },
249   formatDate: function(date){
250     var spl=date.split('-');
251     return(spl[2]+". "+spl[1]+". "+spl[0]);
252   },
253   //renderování celé komponenty
254   render: function() {
255     var dashboardTableComponent = null; //komponenta
          tabulky
256     var pagination = null; //stránkování
257     var addForm = null; //přidávací formulář
258     var editForm = null; //editovací formulář
259     var chartComponent = null; //graf
260     if (this.state.showAddForm==true){ //pokud je stav
          zobrazení formuláře true
261       addForm = <EditForm clickHandler={
            this.handleAddFormRem} type="add" month={
            this.state.month} year={this.state.year} page={
            this.state.page} />;
262     }
263     if (this.state.showEditForm==true){ //pokud je stav
          zobrazení formuláře true
264       editForm = this.state.editForm;
265     }
266     if (this.state.data!=null) { //pokud jsou data načtená
          ajaxem
267       if (this.state.pages>1) { //pokud je více stránek,
            než 1
268
269         //vytvoření pole stránkování
270         var pagesArray = new Array();
```

```
271         for(var i=1;i<=this.state.pages;i++)
272         {
273           pagesArray.push(i);
274         }
275         //výpis stránkování
276         pagination =
277           <ul className="pagination pull-center">
278             {
279               pagesArray.map(function(page) {
280                 return <li key={page} className={(
                      this.state.page == page ? '
                      paginationBtn active' : 'paginationBtn
                      ')}><a onClick={
                      this.handleClickChangePage.bind(this,
                      page)}>{page}</a></li>
281               }, this)
282             }
283           </ul>;
284       }
285     }
286
287
288       $('#categoryChartCanvas').remove();
289       $('.chartCanvasContainer').append('<canvas id="
            categoryChartCanvas" width="300" height="300"></
            canvas>');
290       var ctx = document.getElementById('
            categoryChartCanvas');
291       var colorSet = [ "#FF6384", "#36A2EB", "#FFCE56", "#
            BBBE64", "#EAF0CE", "#C0C5C1", "#7D8491",
            "#443850", "#F786AA", "#A14A76", "#6DCFD6", "#
            B3D3D6", "#562756", "#605760", "#119DAF" ];
292
293       if (ctx!=null && this.state.categoriesData!=null) {
```

```
294         var data = {
295           labels:  this.state.categoriesLabels,
296           datasets: [
297             {
298               data: this.state.categoriesData,
299               backgroundColor: colorSet,
300               hoverBackgroundColor: colorSet
301             }]
302         };
303         var categoryChart = new Chart(ctx, {
304           type: 'pie',
305           data: data,
306           options: {
307             animation:{
308               duration:0
309             }
310           }
311         });
312       }
313
314     var empty=null;
315     if (this.state.data.length==0) empty = <tr><td colspan
          ="6">Nejsou žádné záznamy</td></tr>;
316     return <div>
317         {addForm}{editForm}
318         <form action="" >
319           Období: 
320           <select onChange={this.handleChangeMonth} value
              ={this.state.month} name="selectMonths" id="
              selectMonths" className={"form-control form-
              control-inline"}>
321             <option value="1">leden</option>
322             <option value="2">únor</option>
323             <option value="3">březen</option>
```

```
324             <option value="4">duben</option>
325             <option value="5">květen</option>
326             <option value="6">červen</option>
327             <option value="7">červenec</option>
328             <option value="8">srpen</option>
329             <option value="9">září</option>
330             <option value="10">říjen</option>
331             <option value="11">listopad</option>
332             <option value="12">prosinec</option>
333           </select> 
334           <select onChange={this.handleChangeYear} id="
                 selectYears" name="selectYears" className="
                 form-control form-control-inline">
335             {
336               this.props.years.map(function(year) {
337                 return <option key={year} value={year}>{
                       year}</option>
338               })
339             }
340           </select>
341         </form>
342
343         <div className="col-md-8 dashboardTableContainer">
344           <div className="panel panel-default">
345           <table className="table table-condensed
                 dashboardTable">
346             <thead>
347               <tr>
348                 <th>Datum</th>
349                 <th className={"hidden-xs"}>Kategorie</th>
350                 <th>Popis</th>
351                 <th>Cena</th>
352                 <th></th>
353                 <th></th>
```

```
354                 </tr>
355               </thead>
356               <tbody>
357                 {
358                   this.state.data.map(function(item) {
359                     return (
360                       <tr key={item[0]}>
361                         <td>{this.formatDate(item[2])}</td>
362                         <td className="hidden-xs">{item
                              [5]}</td>
363                         <td>{item[4]}</td>
364                         <td>{item[3]} Kč</td>
365                         <td><span className="glyphicon
                              glyphicon-edit editBtn" onClick={
                              this.handleEditForm.bind(this,
                              item)}></span></td>
366                         <td><span className="glyphicon
                              glyphicon-remove deleteBtn"
                              onClick={
                              this.handleClickDelete.bind(this,
                              item)}></span></td>
367                       </tr>
368                     )
369                 }, this)
370               }
371                 {empty}
372               </tbody>
373             </table>
374           </div>
375             <button className="btn btn-primary pull-right"
                 onClick={this.handleAddForm}>Přidat</button>
376             <div className="clearfix"></div>
377             <nav className="text-center">
378               {pagination}
```

```
379             </nav>
380           </div>
381
382           <div className="col-md-4 visible-md visible-lg
                  chartCanvasContainer"></div>
383         </div>;
384     }
385 });
386
387 //vytvoření pole se zobrazenýma rokama
388 var date = new Date();
389 var year = date.getFullYear();
390 var month = date.getMonth()+1;
391 var years = new Array();
392 for (var i=year;i>=2010;i--){
393   years.push(i);
394 }
395
396 //aktivace komponenty
397 ReactDOM.render(<TableDashboardComponent years={years}
      year={year} month={month} />, document.getElementById('
      tableDashboardComponent'));
```