VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SOFTWARE PRO ONLINE REKLAMACE

SOFTWARE FOR ONLINE WARRANTY CLAIMS

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE — **Bc. PETR POLOLÁNÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19680/2016/xpolol01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Pololáník Petr, Bc.**
Obor: Informační systémy
Téma: **Software pro online reklamace**
Software for Online Warranty Claims
Kategorie: Uživatelská rozhraní

Pokyny:

1. Prostudujte dostupnou literaturu na téma React a příprava software v tomto systému.
2. Zvažte možnosti implementace systému online reklamací v React a diskutujte možné vlastnosti.
3. Implementujte systém online reklamací (klientskou část) a demonstrujte výsledky na vhodném příkladu.
4. Diskutujte možnosti implementace systému na mobilních zařízeních.
5. Vyhodnoťte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynu vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, prof. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá vytvářením moderních webových aplikací. Jsou rozebrané technologie, které se používají při návrhu a implementaci těchto aplikací. Práce se převážně zabývá vytvořením aplikace pro vyřizování reklamací. Aplikace slouží jako služba pro internetové obchody

Abstract

This thesis deals with the theory of creating modern web applications. Various technologies that are commonly used are analyzed. Thesis mainly deals with creating specific application for processing Warranty Claims. The application serves as a service for eshops.

Klíčová slova

SPA, web, react, redux, javascript, reklamace

Keywords

SPA, web, react, redux, javascript, warranty claims

Citace

POLOLÁNÍK, Petr. *Software pro online reklamace*. Brno, 2017. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zemčík Pavel.

Software pro online reklamace

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Prof. Dr. Ing. Pavel Zemčík Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Pololáník
24. května 2017

Poděkování

Chtěl bych poděkovat svému vedoucímu semestrální práce Prof. Dr. Ing. Pavel Zemčík za odborné vedení, za pomoc a rady při zpracování této práce.

Obsah

Kapitola 1

Úvod

V dnešní době existuje mnoho e-shopů a lidé jsou na nich zvyklí nakupovat zboží stále častěji. S každým takovým nákupem vzniká riziko, že zákazník bude chtít zboží vrátit, nebo se porouchá a bude ho muset reklamovat. Oproti nákupu v kamenném obchodě se žádosti o vrácení či reklamace (dále jen žádosti) musí řešit odlišným způsobem. Důvodem je, že ne všechny obchody mají své kamenné pobočky. E-shopy tyto žádosti většinou řeší pomocí série emailu či telefonátů. To může být velice nepříjemné pro obě strany. Některé obchody ovšem nabízí již předpřipravené formuláře na svých stránkách či přikládají tyto formuláře k odeslanému zboží. Existují moduly, které zákazníkům i obchodům usnadňují vyřizování takovýchto žádostí. Tyto moduly jsou ovšem zaměřené na konkrétní systémy, na nichž jsou obchody postaveny.

Cílem této diplomové práce je analyzovat stávající systémy pro správu žádostí a na základě nich pak vytvořit vlastní systém, který by řešil obecnější problémy a mohl být dostupný pro většinu běžných obchodů. Výsledná aplikace by měla umožnit obchodům, aby svým zákazníkům daly možnost snadno reklamovat zboží. Samotné obchody by měly mít k dispozici nástroj pro snadnou a přehlednou správu těchto žádostí. Díky tomu by mohly zákazníci i obchody ušetřit čas. Tato práce neřeší serverovou část, pouze komunikaci s ní.

V první části této práce je popis některých technologií, které se používají při tvorbě webových aplikací. Vzhledem k tomu, že se v rámci diplomové práce řeší klientská část, je tato kapitola zaměřena především na klientské technologie webových aplikací. V této části je popsán převážně jazyk JavaScript, knihovna React a nástroje pro tvorbu uživatelského prostředí. Pro vývoj webových aplikací se dnes běžně používá běhové prostředí NodeJS, kterému je věnovaná část kapitoly. V další kapitole jsou popsány způsoby, jakými reklamace řeší některé větší české obchody. Součástí kapitoly je i seznam a popis modulů, které přidávají funkční prvky pro snadnější vyřizování reklamací. Poté následuje kapitola obsahující zhodnocení předchozích kapitol a seznam požadavků na výslednou aplikaci včetně podrobnějšího popisu uživatelských rolí. Ke konci práce je kapitola popisující implementaci. Ze začátku obsahuje návrh hlavních obrazovek aplikace. Pro lepší porozumění je obsahem i grafické znázornění. Po něm následuje detailnější popis některých implementovaných částí. Poté je ukázané použití aplikace pro ukázkový obchod. Kapitola je zakončena popisem testů, které by bylo vhodné provést před produkčním nasazením aplikace. Některé testy byly provedeny a zhodnoceny. Práce končí závěrem, ve kterém jsou zhodnoceny dosažené výsledky. Součástí jsou i možnosti rozšíření aplikace.

Kapitola 2

Technologie

Na začátku první kapitoly jsou popsány dva koncepty, na kterých lze stavět webové aplikace 2.1. Následovat bude podkapitola 2.2 pojednávající o tom, jak spolu komunikuje server a klient. V podkapitole 2.3 je popsána platforma NodeJS, kterou je možné využít při tvorbě či provozu webové aplikace. Dále obsahuje popis užitečných modulů. Důležitou součástí každé aplikace je její grafické uživatelské prostředí. Věci s tímto spojené jsou popsané v části 2.4. V závěru kapitoly je podrobnější pohled na technologii React a věci s ní úzce spojené 2.5. Práce není „encyklopedickým přehledem“ daného oboru. Některé zmíněné technologie jsou nové a není pro ně dostatek literatury. Z tohoto důvodů jsou pro některé zdroje použity internetové stránky.

2.1 Single Page Application (SPA)

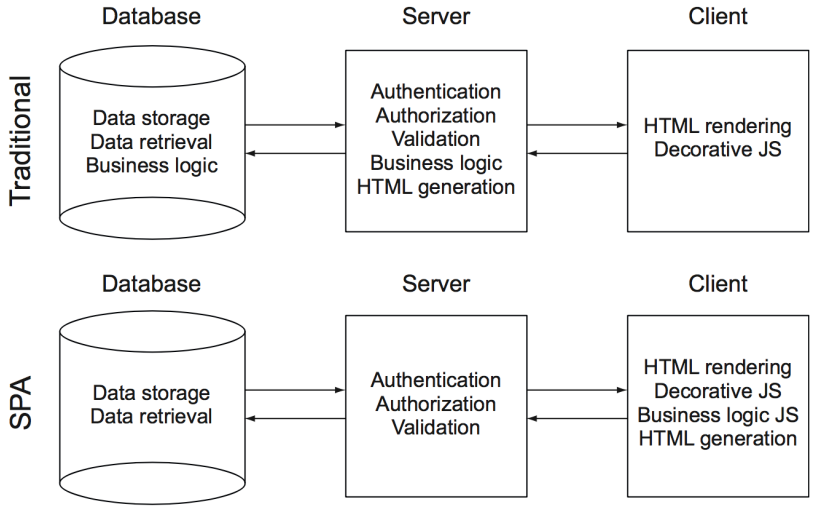
Single page application (SPA) Procházení tradičních stránek je velice pomalé. Je to dáno tím, že když uživatel klikne na odkaz na stránce, prohlížeč musí kontaktovat server. Poté stáhnout celou HTML strukturu stránky a znovu ji vykreslit. I když uživatel potřebuje načíst jen určitou změnu, přesto se mu stahuje vše. Tradiční webové frameworky jsou totiž zaměřené na serverovou část. Ta vytváří statickou strukturu stránky. kterou odesílá klientovi. Takovýto způsob ubírá na komfortu prohlížení webu.

V posledních letech se začínají více používat tzv. Single Page Application, dále jen SPA. Koncept je postaven na tom, že už se server nestará o generování HTML struktury, ale pouze posílá klientovi data, které si zažádá. Generování tedy provádí klient, konkrétně JavaScript, který komunikuje se serverem a vyměňuje si s ním data. Při procházení stránky tedy nedochází ke stahování velkého objemu dat, ale pouze toho, co je zrovna potřeba a co klient ještě nemá. Procházení webu je tedy obzvláště rychlé. Oproti tradičnímu konceptu je většina logiky přenesena na klientský JavaScript. Zároveň jsou obě strany oddělené a je tedy možné jednu ze stran nahradit novou, aniž by bylo potřeba upravovat druhou stranu. Pro komunikaci mezi klientem a serverem je možné využít klasické HTTP dotazy pomocí ajaxu a nebo využít webové sockety. Rozdíly mezi těmito dvěma koncepty je znázorněn na obrázku 2.1.

SPA se podobá klasickým nativním aplikacím. Při načítání mohou na patřičném místě zobrazit oznámení. Dokonce mohou omezeně pracovat v offline režimu.

Důvody, proč se SPA začíná více používat až v posledních letech, jsou následující:

- vznikly nové technologie (HTML5, SVG a CSS3),



Obrázek 2.1: Client Server vs SPA, převzato z [14].

- dříve byl JavaScript nestabilní a choval se v různých prohlížečích odlišně, to už dnes neplatí,
- webový prohlížeč se stal nejpoužívanější aplikací. Mnoho lidí ho má téměř neustále otevřený,
- JavaScript je použitelný na všech platformách. Tím je myšleno i na mobilních telefonech a tabletech,
- interpret JavaScriptu ve webových prohlížečích je rychlejší, než byl dříve.

Technologie jako je Flash nebo Java applets se pomalu přestávají využívat. Důvodů je několik:

- nedokáží využívat všechny prostředky,
- liší se vzhledově i funkčně, např. formulářové prvky, nebo klik pravého tlačítka,
- často se v nich objevují chyby.

Jak již bylo zmíněno, role webového serveru se mění. Přestože se většina logiky přenesla na klienta, server je stále potřeba. Důvody jsou autentizace, autorizace, validace dat, umístění dat a synchronizace [14].

2.2 Komunikace klienta se serverem

V předchozí podkapitole 2.1 byly probrány dva rozdílné koncepty webových aplikací. Tato sekce bude probírat, jakými způsoby lze provádět komunikaci mezi klientem a serverem u SPA. Zprávy, které si tyto dvě strany vyměňují, jsou často ve formátu JSON nebo XML.

Prvním způsobem je REST (Representational State Transfer) architektura určená pro distribuované prostředí. Princip zpočívá v tom, že existují různé zdroje, které mají vlastní URI. Klient může k těmto zdrojům přistoupit pomocí 4 metod a pracovat s nimi. Metody se označují jako CRUD (create/read/update/delete). Tabulka 2.1 popisuje jejich význam.

HTTP název	Akce	Analogická databázová operace
POST	Create (or append)	insert
GET	Retrieve	select
PUT	Update (or create)	update
DELETE	Delete	delete

Tabulka 2.1: REST metody, převzato z [15].

Tyto metody jsou volány pomocí HTTP dotazů s metodou dle tabulky. Komunikace probíhá tak, že se klient pomocí ajaxu[1] dotazuje a server mu na dotaz odpovídá. Nevýhodou této komunikace je to, že pokud by chtěl klient stále aktuální data, musí se neustále dotazovat serveru, zda nedošlo k nějaké změně [15].

Druhým příkladem komunikace mohou být Web Sockety, které přišly se standardem HTML5. Ajax je limitován tím, že komunikaci musí vždy vyvolávat klient, nikoliv server sám. Web Sockety umožňují vzájemnou komunikaci mezi klientem a serverem podobně jako klasické sockety. Spojení, které se naváže, může běžet nepřetržitě a mohou se přes něj posílat zprávy v reálném čase oběma směry. Zprávy, které si vyměňují, jsou pouhé řetězce a je na aplikacích, aby si zprávu správně zpracovaly. Websockety v některých starších verzích prohlížečů nejsou podporované [2].

2.3 NodeJS

Platforma NodeJS je běhové prostředí pro rychlé vytváření a běh škálovatelných síťových aplikací. Platforma je postavena na V8 (JavaScript engine) od Googlu [3].

NodeJS využívá událostmi řízenou architekturu a neblokující vstup/výstupní model. Díky tomu je vhodný pro real-time aplikace, které běží distribuovaně. Aplikace se píší v jazyce JavaScript stejně jako klientská část webových aplikací. Výhodou pro vývojáře zvolit si NodeJS jako platformu pro server je, že může používat jeden programovací jazyk pro obě strany. NodeJS drží krok s nejnovějším standardem ECMAScriptu. Díky tomu lze při vývoji využívat moderní konstrukce jazyka, které by některé prohlížeče nemuseli podporovat.

Prostředí lze využít pro běh lokálních aplikací nebo pro vytvoření serverové aplikace. Podporuje TCP komunikaci, sockety, práci se soubory a lze do něj dopsat i vlastní moduly v jazyce C++. Jazyk JavaScript je používaný i v několika databázích NoSQL (například MongoDB). S takovýmito databázemi lze tedy snadno komunikovat a pracovat. Postupem času se NodeJS stal nástrojem, určeným pro běh různých utilit k usnadnění práce.

Pro plaformu existuje mnoho modulů, které jsou dostupné přes balíčkový manager npm, jenž je obsažen v základní instalaci NodeJS. V době psaní této práce je v repozitáři npm 395 028 balíčků. Seznam balíčků je dostupný na https://www.npmjs.com/. Balíčky lze nainstalovat pomocí příkazů `npm install [název_balíčku]`. Vhodnější je ale vytvořit soubor `package.json`, který obsahuje základní informace o projektu včetně seznamu balíčků ,na kterých je závislý. Poté stačí pouze příkaz `npm install` a nainstalují se veškeré závislosti. Stažené balíčky se uloží do složky `node_modules`, která se vytvoří v aktuálním adresáří.

[1] Ajax (asynchronní JavaScript a XML) je technogolie pro komunikaci klientské aplikace v jazyce JavaScripte se serverem

[2] Popis API https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[3] V8 JavaScript engine je vytvořený v rámci The Chromium Project pro webový prohlížeč Google Chrome

Každý balíček obsahuje svoji verzi. Díky tomu lze snadno verzovat a zavádět závislosti na konkrétní verze. Stejně jako stahovat balíčky je možné je i snadno do repozitáře přidat. Jejich konfigurace se řídí zase pomocí souboru `package.json` [13].

Pro NodeJS existují balíčky, které mají za cíl usnadnit a zlepšit vývoj. Jejich úkony jsou kupříkladu následující:

- spojování a optimalizování zdrojového kódu,
- provedení preprocesingu a postprocesingu,
- zmenšování obrázků,
- automatické hlídání změn.

Díky těmto věcem je proces vytváření webových aplikaci odlišný než byl dříve. Aplikace může být rozdělena do několika souborů, které na sebe vzájemně odkazují. Podobně jako je tomu například v jazyce C++.

Webpack

Webpack je modul, který běží nad NodeJS prostředím. Slouží pro práci se zdrojovým kódem. Dokáže spojit soubory aplikace napsané v JavaScriptu dle definovaných vlastností a vytvořit z nich jeden soubor. Zároveň je schopen do výsledného souboru přidat i balíčky z npm. Webpack se spustí nad projektem, u kterého hlídá změny. Pokud vývojář upraví nějaký soubor, webpack to rozpozná a provede patřičné operace. Při vytváření a testování webové aplikace je vhodné, aby aplikace fungovala lokálně. Tomu napomáhá vývojový server (součást webpacku). Webpack se konfiguruje přes speciální konfigurační soubor a nabízí mnoho pluginů pro různé úkony. Například minifikace souborů, scalování obrázků, SASS, LESS. Spouští se pomocí příkazového řádku. [2].

Babel

Jazyk JavaScript se stále vyvíjí a ne všechny prohlížeče podporují veškeré jeho funkce. Aby vývojáři měli zpětnou podporu starších prohlížečů, a nemuseli se bát použít nejnovější věci z jazyka, vznikl Babel. Jedná se o nástroj, který je schopen převéset novou verzi jazyka do staré bez ztráty funkčnosti. Tím lze při vývoji používat mnoho nových konstrukcí jazyka. Převod lze udělat manuálně a nebo využít některého z modulů pro prostředí jako je webpack či jiné. Babel lze rozšířit o pluginy, které přidávají další možné konstrukce[17].

2.4 Uživatelské prostředí

Uživatelské prostředí webové stránky se převážně skládá ze 3 částí: HTML, CSS a JavaScript. Všechny tyto zdroje jsou uloženy na serverovém počítači a pomocí protokolu HTTP jsou přeneseny do klientského prohlížeče uživatele. Při vytváření uživatelského prostředí je nutné myslet na to, že se výsledný obsah může mírně lišit dle typu prohlížeče. Každý prohlížeč je na tom s podporou nejnovějších věcí ze standardu odlišně. Mnoho lidí navíc používá zastarelé a neauktualizované prohlížeče. Ani to že je prohložeč v nejaktuálnější verzi ještě neznamená, že podporuje vše. Tato fakta je zapotřeba mít vždy napaměti a výsledné prostředí testovat na více platformách a ve více verzích. Na webové stránce

http://caniuse.com/ si lze ověřit, jak je daná věc podporována napřič různymy prohlížeči a jejich verzemi [3].

HTML je značkovací jazyk, který popisuje strukturu webové stánky neboli dokumentu. K popisu využívá elementy, jenž se dělí na párové a nepárové dle toho, zda v nich může být vnořený další element či text. Každý element je určitého typu, který popisuje jeho charakter a vizuální podobu. Příkladem elementu může být nadpis, odstavec, seznam, obrázek a další. Elementům lze přiřazovat atributy, které jim přidávají více informací. Atribut se skládá ze jména a hodnoty. Například pro element *obrázek* existuje atribut `src`, jenž udává zdroj obrázku. HTML má několik verzí, přičemž poslední verzí je HTML5 [3].

CSS umožňuje vytvářet pravidla, která určují, jak budou dané elementy na stránce vypadat. Pravidla se skládají ze selektoru a části pro deklaraci. K deklaraci lze přiřadit více selektorů. Deklarace popisuje vizuální styl elementu pomocí množiny párů vlastnost a hodnota. Selektorem lze vybrat prvek například určitého typu, určitého zanoření a jiné. CSS lze zapisovat přímo do HTML struktury a to do speciálního elementu `<style>`. Druhou možností je mít vlastní soubor CSS a na ten se odkázat v HTML pomocí elementu `<link>`. Poslední verze je CSS3. Oproti předchozí verzi přidává mimo jiné možnost vytvářet animace. Při psaní stylů se dají využít i Media Queries. Ta přidávají více možností jako například vybírat některá pravdila na základě rozlišení obrazovky či orientaci displeje[4][3].

JavaScript je programovací jazyk, který přidává interakci k webovým stránkám a dokáže dynamicky měnit jeho obsah. Lze s ním vyvolávat akce na různé události jako je například kliknutí na tlačítko, přejetí myší přes nějaký element atd. Jeho časté použití je ve slideshow[5], validace formulářů[6], filtrování dat na stránce nebo stahování a zobrazování dodatečného obsahu. JavaScript má přístup k HTML obsahu pomocí objektu DOM (Document Object Model), což je stromová struktura objektů, která dovoluje přistupovat ke stromové struktuře HTML elementů. Pomocí JavaScriptu a DOM lze tedy veškerý HTML obsah modifikovat. Kód jazyka lze vložit stejně jako CSS dvěma již zmíněnými způsoby [4].

Reakce na změny

V posledních letech stále více lidí navštěvuje webové stránky pomocí svého telefonu. Společnost Gemius poskytuje statistiky návštěvnosti webových stránek v tuzemsku z několika pohledů. Jedním z nich je podíl návštěvnosti dle operačního systému, jak ukazuje 2.2. Lze vidět, že podíl operačních systémů pro mobilní zařízení každý rok roste a v roce 2016 dosahoval pětiny veškerého naměřeného přístupu. V některých státech tento podíl bývá nadpoloviční. Na tento fakt musí reagovat webdesigneří a přizpůsobovat tomu webové rozhraní. Z toho důvodu by se web, který by měl být dostupný pro všechny, neměl navrhovat ve fixních rozměrech, ale měl by být responzivní[7]. Avšak je i jiný způsob, jak se řeší web pro mobilní zařízení a to vytvořením separátního webu určeného pouze pro mobilní zařízení. Tento způsob je ale pro návštěvníky i pro provozovatele webu nevhodný a postupně se přestává používat.

[4]Orientací displeje se myslí, zda je displej otočen do určité polohy. Pokud je šířka větší než výška, jedná se o landscape v opačném případě se poloha nazývá portrait

[5]Slideshow je sekvence automaticky se měnících obrázků na stránce. JavaScript zde provádí výměnu obrázku po určitém čase nebo po kliknutí na patřičné tlačítko

[6]Validace je ověření zda uživatel vyplnil správně dané vstupní pole, například zda je správně naformátované telefonní číslo. Javascript uživateli může u špatně vyplněného pole zobrazit patřičnou hlášku a neumožnit mu odeslat formulář

[7]Responzivní web je takový, který se přizpůsobí šířce displeje tak, aby byl použitelný

Operační systém	**28.11-04.11.2016**	**30.11-06.12.2015**	**01.12-07.12.2014**
Windows 7	29.43 %	41,09 %	49,69 %
Windows 10	27,64 %	10,60 %	0 %
Android	19,11 %	15,37 %	10,27 %
Windows 8.1	6,10 %	11,94 %	10,56 %
Windows XP	5,68 %	9,19 %	15,36 %
iOS	4,37 %	3,47 %	2,83 %
Linux	2,29 %	1,53 %	1,42 %
Windows Vista	2,00 %	3,17 %	4,72 %
Mac OS X	1,24 %	1,15 %	1,04 %
Windows 8	0,94 %	1,58 %	3,25 %
Windows Phone 8.1	0,65 %	0,54 %	0,00 %
Windows Phone 10	0,32 %	0,02%	0,00 %

Tabulka 2.2: Statistika návštěvy webových stránek dle operačního systému v jednotlivých obdobích. Zdrojem dat je web rankings.cz.

Jazyk CSS3 umožňuje vytváření responzivních webů pomocí svých vlastností a Media Queries. Optimalizovat web pro všechny prohlížeče je velice těžké a ne vždy se to dělá. Vhodné je si udělat vlastní statistiky návštěvnosti a na základě nich optimalizovat. V posledních letech se lze setkat s pojmem retina displej. Toto označení znamená, že displej má velice jemné rozlišení (vysoké rozlišení vzhledem k úhlopříčce). Webové prohlížeče na takovýchto zařízeních nezobrazují stránky v pravém rozlišení, ale přepočítávají rozměry. V opačném případě by byl web příliš malý a nečitelný. Tomuto přepočtu se říká device-pixel-ratio, což udává v jakém poměru je reálný pixel displeje a softwarový pixel. Je ale třeba počítat s tím, že roztáhnutí bitmapových obrázků poškodí jejich kvalitu. Řešením je použití vektorové grafiky nebo mít obrázek ve větším rozlišení, než v jakém ho chceme zobrazit. U obrázků s větším rozlišením se zvětšuje i jejich datový objem. HTML a CSS nabízí nástroje, které umožňují posílat různé obrázky dle toho, jaký je device-pixel-ratio. Pro vektorové obrázky existuje formát SVG, který lze přímo zapisovat i do HTML.

V dřívějších dobách se pro grafiku na webu využívaly statické obrázky. Důvodem bylo to, že zde nebyl požadavek na responzivitu. V dnešní době je tomu ale naopak a webdesignéři se snaží co nejvíce věcí udělat pomocí HTML, CSS a SVG. K tomu přispívá i mnoho nástrojů pro grafiky, které s tímto faktem počítají a dovolují generovat grafiku přímo jako zdrojový kód. Framework Bootstrap zase přináší možnost vytvářet webové rozhraní pomocí předpřipravených komonent a tudíž nepotřebují grafika či webdesignéra[12].

CSS preprocesory a postprocesory

Jazyk CSS je jednoduchý a neumožňuje zapisovat složitější struktury či lépe sdružovat kód. Psaní větších stylů, tak může být nepřehledné a obtížně udržovatelné. Na to reagovala komunita vývojářů a vytvořila preprocesory[8] SASS a LESS. Tyto preprocesory přidávají více možností pro vytváření stylů a snaží se řešit nejen výše uvedené potíže. Preprocesing probíhá na straně vývojáře. Uživatelům je již poskytován zpracovaný kód. Nevýhodou preprocesorů je, že mohou být někdy až moc silné a dovolí zapsat kód příliš složitě. Zmíněné preprocesory se sice liší, ale i tak jsou pro ně některé následující vlastnosti společné.

[8]Preprocesor je program, který předzpracovává vstup zdrojového textu, před finálním zpracováním cílového programu

- Proměnné - mají zde podobný význam jako v jiných programovacích jazycích. Do vlastností k více pravidlům lze přiřadit proměnnou a její hodnotu pak mít pouze na jednom místě. Například framework Bootstrap lze tímto způsobem celý konfigurovat (například měnit hlavní barvu). Proměnné lze využívat i na více místech než jen jako hodnoty.
- Zanořování je vlastnost kterou lze do sebe zanořovat více pravidel, což silně usnadňuje čitelnost kódu a umožňuje lepší organizaci.
- Mixiny pomáhají s opakujícím se kódem. Lze jim dát parametr a zavolat je na určitém místě, kde vygenerují kód
- Import umožňuje rozdělit kód do více souborů, které se po preprocesingu spojí do jednoho.

V následující ukázce jde vidět převod kódu ze SASS do CSS. V ukázce je použita proměnná a zanoření.

```
/* SASS */
@brand-primary: red;
.container {
   background-color: @brand-primary;
   .item {
      color: @brand-primary
   }
}

/* CSS */
.container {
   background-color: red;
}
.container .item {
   color: red;
}
```

Postprocessing doplňuje preprocesing, ale je možné že ho v budoucnu úplně nahradí. Jak postupně prohlížeče doimplementovávají různé nové vlastnosti do CSS, někdy k nim přidávají prefixy. Důvod je, že ve finální podobě se daná vlastnost může lišit od stávající. Při psaní některých vlastnostní je nutné přidat i prefixkové varianty. K tomu pomáhá autoprefixer, který funguje právě jako postprocessing. Vývojář tedy napíše vlastnost pouze jednou a to bez prefixu. Poté zvolí pro jaké prohložeče chce přidat prefixy a autoprefixer to za něj udělá [12][7].

Twitter Bootstrap

V roce 2011 Mark Otto a Jacob Thornton, zaměstnanci v Twitteru, uvolnili open source framework zvaný Bootstrap. Jedná se o sadu stylů a JavaScriptů, které pomáhají při vytváření tzv. frontendu. Obsahuje výchozí styly pro standardní prvky na stránce jako jsou formuláře, tlačítka, menu, a jiné. Díky tomu lze ušetřit spoustu času a začít rychle vytvářet stránku, která je funkční a estetická. Každý prohlížeč může mít výchozí styly elementů nastavené různě. Bootstrap obsahuje pro základní elementy předchystané styly, tak aby vizuálně vypadaly ve všech prohlížečích stejně.

Vývojář není nucen používat vše, ale může si vybrat jen některé prvky, které použije, nebo styly upraví podle sebe. Bootstrap je responzivní a vytvořen pro styl mobile-first (začíná se od mobilní verze webu a pokračuje do desktopové). Implementovat bootstrap je snadné. Stačí přidat 2 odkazy na externí zdroje do hlavičky stránky. V případě že si chce uživatel bootstrap upravit nebo si z něj vytáhnout jen to, co potřebuje, může si stáhnout nezkompilovanou verzi a kompilaci si provádět sám.

Práce s bootstrapem je založená na přidělování tříd elementům. Obyčejné tlačítko bez boostrapu lze zapsat jako `<button>Tlačítko</button>`,). Pro přidání stylu pak stačí napsat `<button class="btn btn-default">Tlačítko</button>`. Třída btn-default je rozšíření třídy btn a přidává možnost zvolit charakter tlačítka a tím ho vizuálně odlišit. Lze zvolit i btn-success, btn-warning, btn-info, btn-danger a btn-link. Na obrázku 2.2 je ukázáno jak se od sebe liší tlačítka s odlišným charakterem. Kromě tlačítek je možné charakter volit i u dalších prvků.

Obrázek 2.2: Bootstrap tlačítka, převzato z [16].

Základní rozdělení stránky do jednotlivých bloků se provádí pomocí tzv. grid systému. Ten je založený na tom, že rozděluje stránku na řádky a ty následně na sloupce. Sloupce mohou mít velikosti šířek závislé na velikosti okna stránky. Díky tomu lze vytvořit responsivní layout. Vytváření takovéhoto layoutu probíhá pomocí přidávání tříd elementům, které reprezentují řádek, či sloupec. U každého sloupce se definuje, jak bude vypadat pro různé šířky. Ve výchozím stavu bootstrap rozděluje stránku do 12 sloupců. Pokud bychom chtěli, aby byl daný řádek rozdělen na 2 sloupce o stejné šířce, nastavím jim velikost 6 sloupců (12/2).

Zlomové body (break points), určují hranici mezi zařízeními, pro které se má layout stránky měnit. Určují při jaké šířce okna se má stránka interpretovat jako velké zařízení, malé zařízení, table, či mobilní telefon.

U struktury stránky se určuje i to zda má fluidní design, či nikoliv. Tímto designem je myšleno, že řádky se roztahují na plnou možnou šířku. V případě, že stránka není fluidní, tak se drží maximální možné šířky. Těchto pevných šířek je několik a z důvodů responsivity. Pro každý typ zařízení je tedy jiná pevná šířka. Při zmenšování okna, se šířka skokově zmenšuje, vždy když se přesáhne zlomový bod.

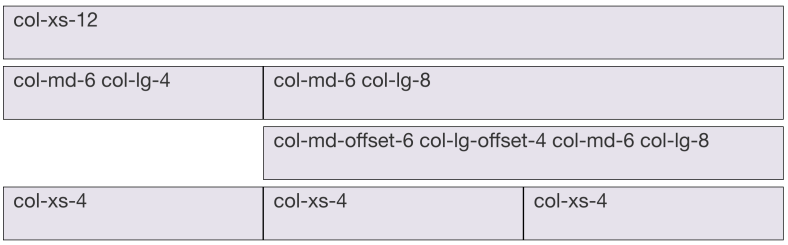
Následující html kód ukazuje příklad možného layoutu. Třídy `row` reprezentují řádky. Ty mají s sobě zanořené elementy, jejichž třídy mají prefix `col-`. Tyto třídy reprezentují sloupce. Každý sloupec může obsahovat více tříd. Kromě šířky sloupce je možné definovat i odsazení zleva.

```
<div class="container-fluid">
  <div class="row">
    <div class="col-xs-12 red"></div>
   </div>
   <div class="row">
    <div class="col-md-6 col-lg-4 red"> </div>
    <div class="col-md-6 col-lg-8 red"></div>
  </div>
  <div class="row">
    <div class="col-md-offset-6 col-lg-offset-4 col-md-6 col-lg-8 red"></div>
```

```
    </div>
    <div class="row">
      <div class="col-xs-4 red"></div>
      <div class="col-xs-4 red"></div>
      <div class="col-xs-4 red"></div>
    </div>
</div>
```

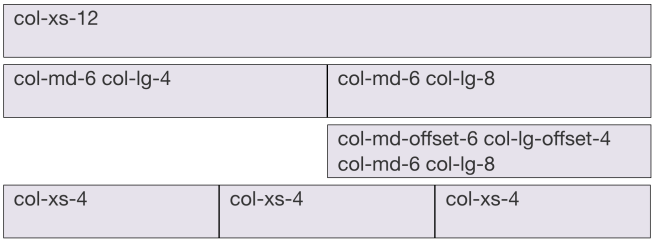
Listing 2.1: Some Java code

V nejvyšší úrovni lze vidět třídu `container-fluid`, která nastavuje fluidní charakter. Kdybychom chtěli pevné šířky, tato třída by se nahradila třídou `container`. Následující obrázky 2.3, 2.4, 2.5 znázorňují, jak je daný kód interpretován na zařízeních s odlišnou šířkou.



Obrázek 2.3: Bootstrap grid systém na desktopu.

Obrázek 2.3 znázorňuje velká zařízení. Poměr šířky sloupců ve druhém řádku je 1:2, to je dáno tím, že se aplikovali třídy s prefixem `col-lg-`. Přičemž na obrázku 2.4 je poměr těchto sloupců 1:1, důvodem, je že už se ignoruje nastavení pro velká zařízení a tudiž se aplikuje rozvržení nastavené pomocí tříd s prefixem `col-md-`.



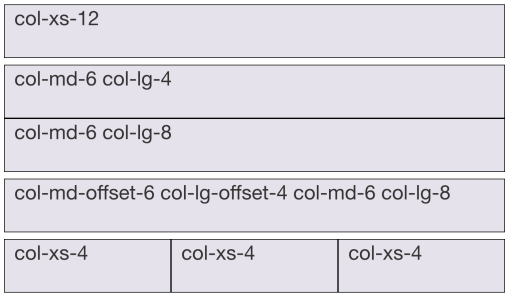
Obrázek 2.4: Bootstrap grid systém na tabletu

Na obrázku 2.5 lze vidět podobu rozvržení na mobilních zařízeních. Pouze na posledním řádku je více sloupců, neboť jsou definované pro malá zařízení.

V tabulce 2.3 je přehled zlomových bodů a chování sloupců pro různá zařízení.

Kromě základních elementů je možnost využít i komplexnější, které mohou být z části složeny ze základních. Tyto elementy mohou pracovat i s bootstrap JavaScriptem, ke kterému je ale nutné přidat knihovnu jQuery. Příkladem komplexního elementu je dialogové okno, rozbalovací menu či obrázková slideshow.

Bootstrap ve svém balíčku nabízí 140 základních ikon, které lze snadno použít na stránce. Ikony nejsou obrázky, ale speciální písmo. Díky tomu jsou ikony stejně jako písmo vektorové a lze je tedy zvětšovat bez ztráty kvality obrazu.



Obrázek 2.5: Bootstrap grid systém na telefonu.

Tabulka 2.3: Bootstrap grid systém.

	Extra malá	**Malá**	**Střední**	**Velká**
Typ zařízení	telefonu	tablety	desktopy	desktopy
Maximální šířka	<768 px	>= 768 px	>=992 px	>=1200 px
Šířka kontejneru	Žádná	750	970	1170
Prefix tříd	.col-xs-	.col-sm-	.col-md-	.col-lg-
Šířka mezert	Auto	~62px	~81px	~81px
Chování gridu	Celou dobu horizontální	Pod sebou, potom horizontálně		

Při správné práci s bootstrapem lze vytvořit web, jehož kód je kvalitnější a čitelnější. Ovšem ne ve všech případech je vhodné framework používat [12][16].

Flexbox

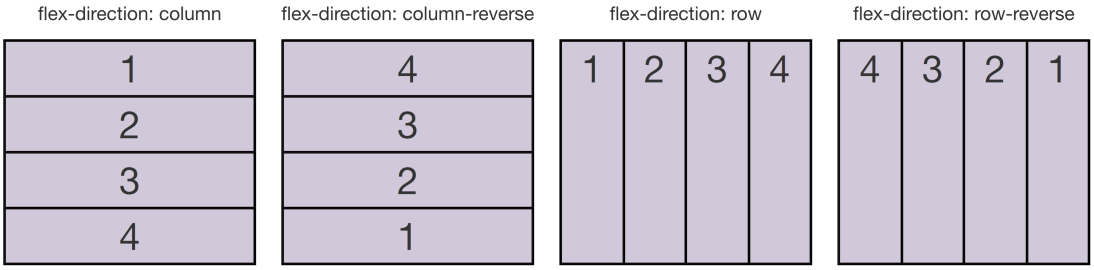
V dřívější době byl nutné pro složitější layout webové stránky využívat prvky a vlastnosti, které k tomu nebyly navržené, ale dokázali problém vyřešit. Příkladem může být layout pomocí tabulek, nebo používání JavaScriptu na dopočítávání a upravování velikosti či pozice elementu. Flexbox je způsob jak vytvářet layout, který je pružnější, aniž by bylo nutné používat JavaScript. Jedná se o první layoutovací nástroj. Je plně podporován nejnovějšími verzemi nejpoužívanějších prohlížečů, kromě prohlížeče Internet Explorer [9]. Práce s ním je založená na CSS vlastnostech. Rodičovský elementu se nastaví vlastnost `display` na hodnotu `flex`. Přímí potomci jsou automaticky prvky. Dalšími CSS vlastnostmi se layout přizpůsobuje požadavkům. Pomocí flexboxu můžeme vyřešit následující problémy:

- totožná výška sloupců v jednom řádku
- šířky položek v poměru
- pořadí položek
- směr jakým se položky vyskládají
- jakým způsobem se zalamují
- zarovnání na osách
- smršťování

[9] Přehled podpory flexbox v prohlížečích http://caniuse.com#feat=flexbox

- zvětšování
- a jiné

Na následujícím obrázku 2.6 je znázorněné, jak se mění pořadí položek uvnitř kontejneru pomocí vlastnosti *flex-direction*. Tato vlastnost může nabývat hodnotu *row*, *row-reverse*, *column*, nebo *column-reverse*. Výchozí hodnotou je *row*. Položky je tedy možné seřadit vedle sebe, pod sebe, či tyto možnosti mít v opačném pořadí.



Obrázek 2.6: Flexbox flex-direction.

Flexbox je vhodný především pro vnitřní komponenty na stránce, pro hlavní layout stránky je určený Grid Layout.

Fonty

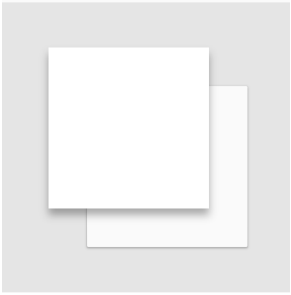
Prohlížeče dnes podporují přidání vlastních fontů. Buď si lze přidat vlastní a nebo využít jednu z cloudových služeb. Přidávat si vlastní font může být náročné, neboť každý prohlížeč podporuje různé formáty. Využívání cloudových služeb je dnes běžné. Mezi takové poskytovatele patří například Google. Jeho databáze fontů je k dispozici na adrese https://fonts.google.com/. Veškeré fonty jsou zdarma a lze si vybrat z velké škály. Následné přidání do webové stránky je pouhé odkázání se na externí zdroj, ať už pomocí CSS nebo HTML. Odkaz je potřeba si vygenerovat. Lze si vybrat různé řezy písma s různými jazyky. Je nutné si dát pozor na to, že ne všechny fonty podporují české znaky [12].

V předchozí kapitole 2.4 byla zmínka o fontech, které mají znaky ikon. Na internetu lze najít spoustu služeb, které tyto fonty nabízí podobným způsobem jako Google. Jedním z nich je https://icomoon.io/. Nabízí mnoho fontů ať už zdarma nebo placených.

Material Design

Material Design je návrhový jazyk vytvořen společností Google a je určen pro tvorbu uživatelského prostředí. Je založen na klasických principech z reálného života jako je papír a inkoust. Snaží se působit tak, jak jsou lidé v běžném životě zvyklí. Prostředí ve kterém pracuje je 3D prostor osvícený zdrojem světla. Světlo vytváří mezi objekty stíny a ty definují hranice jednotlivých částí. Síla stínu určuje v jaké výšce je daný objekt. Čím silnější je stín, tím je objekt výš, takže blíž k uživateli. Příkladem může být seznam, u kterého aktuálně vybraný prvek má nejsilnější stín. Jazyk definuje možnosti jednotlivých objektů, jejich transformace a animace. Material Design využívá Google ve svém operačním systému Android a ve svých aplikacích napříč různými platformami. Existuje mnoho frameworku a knihoven, který s implementací pomáhají. Ukázka hranice prvků je na obrázku 2.7.

Objekty v uživatelském prostředí se nesmí jen tak objevit, ale je potřeba pomocí animace a transformace ukázat, jak se tam objevili (například přiletí z jedné ze stran)[10][6].



Obrázek 2.7: Stíny definují hranice objektů, převzato z [6].

Pattern Lab

Pattern Lab je nástroj, který slouží k návrhu, testování a prezentaci atomického designu. Princip atomického designu je v rozdělení prvků aplikace do kategorií dle úrovní jejich použití. Pattern lab k rozdělení využívá chemickou terminologii:

- **Atomy** základní stavební prvky (tlačítka),
- **Molekuly** kombinace základních prvků (tlačítko s inputem),
- **Organizmy** seskupení molekul (hlavička stránky),
- **Šablony** obsah stránky,
- **Stránky** konkrétní stránka.

Vytváření designu je pouze skládání jednotlivých prvků podobně jako kostky stavebnice Lega. Celkový návrh a testování GUI je pak pohodlnější [11].

2.5 React

React je populární knihovna pro vytváření uživatelského rozhraní. Za jeho vytvořením stojí společnost Facebook a Instagram. Důvod pro jeho vytvoření byl, že Facebook potřeboval vyřešit problémy spojené s vytvářeném webových aplikací ve vetším měřítku. React byl uvolněn jako open source v roce 2013 a ze začátku byl kritizován z důvodů jeho unikátnosti. Tým reactu poté napsal tutoriál s názvem “Why react”, ve kterém v několika minutách představuje React na konkrétním problému. Tím chtěl vývojáře nejprve seznámit s tvorbou pomocí Reactu, než o něm začnou více přemýšlet.

React je malá knihovna, která neobsahuje veškeré věci, které jsou potřeba při tvorbě aplikací. Pro vývoj aplikací je vhodné používat další nástroje jako je například Webpack, Browserify. Je vyžadován pre-processing, neboť se zdrojový kód píše v jazycích, které je potřeba následně kompilovat kvůli podpoře většiny prohlížečů a jazyka JSX. React ve svých webových aplikacích využívá například Facebook, Airbnb, New York Times, Instagram a mnoho dalších. To ukazuje, že je React opravdu dostatečný a stabilní. Pomocí Reactu lze navíc psát i nativní mobilní aplikace pro platformy iOS a Android. K Reactu Facebook vytvořil i Flux, který má na starost datový model. Flux se moc neujal a na jeho myšlence byl postaven Redux.

React je knihnovna nikoliv framework, i když jsou s ni frameworky často srovnávány. React lze použít s frameworky jako je Angular nebo Ember. Frameworky bývají větší než

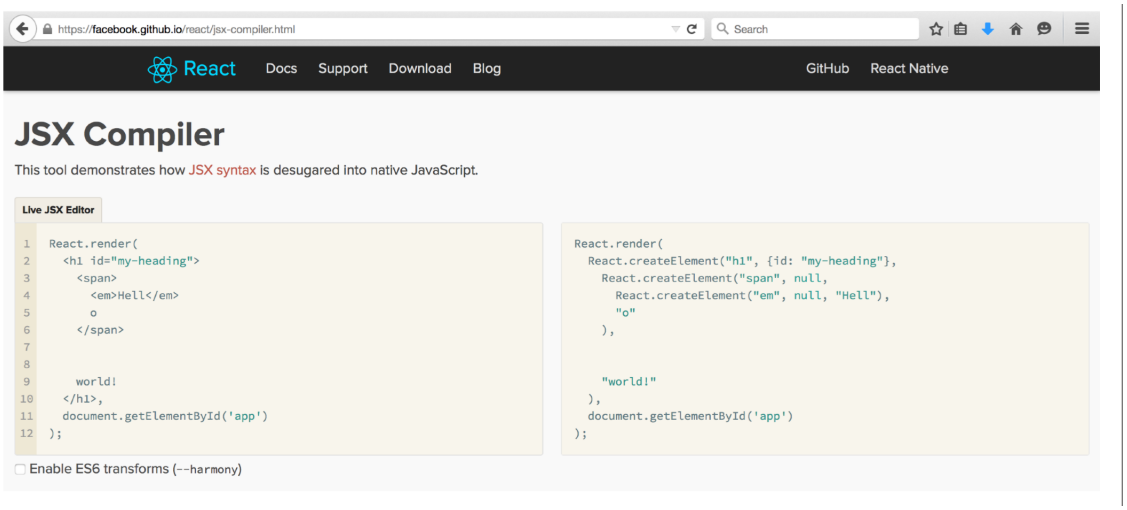
knihovny a obsahuji všechny nezbytné části k tomu, aby šlo vytvořit komplexní aplikaci. React neobsahuje žádné REST nástroje pro HTTP dotazy. Je možné použít téměř jakékoliv knihovny nebo návrhové vzory a k nim přidat React jako view vrstvu.

Okolo Reactu existuje ekosystém populárních knihoven a návrhových vzorů, které se při vytváření aplikací využívají. Je možné si vybrat z velké škály. Navíc existuje spousta startovacích balíčků, které obsahuji všechny nezbytné části. Ovšem je možné psát aplikace pouze pomocí Reactu. Startovací balíčky někdy obsahují obrovské množství věcí, které nejsou potřeba.

Psaní aplikace v Reactu znamená vytváření a propojování komponent. Celek aplikace se poté skládá z několika komponent poskládaných do stromové struktury. Komponentou je objekt, který na základě vstupu a svého vnitřního stavu (pokud ho má) vykresluje obsah. Každá komponenta má tedy určitou zodpovědnost. Díky tomu lze nad jednotlivými částmi aplikace uvažovat odděleně. Komunikace mezi komponentami v zásadě probíhají pomocí takzvaných properties. Na komponenty je možné se dívat jako na elementy v HTML, jejichž vlastnosti jsou dané nastavením atributů [2].

JSX

Pomocí jazyka JSX lze komponenty zapisovat podobně jako HTML kód. JSX je rozšíření JavaScriptu, které se před použitím převede do klasického JavaScriptu. Toto rozšíření silně usnadňuje a zpřehledňuje zápis. Tento převod je součástí již zmíněného balíku Babel. Na 2.8 lze vidět, jak vypadá převod kódů z JSX do JavaScriptu.



Obrázek 2.8: Převod JSX do JavaScriptu, zdroj [9].

Následné poskládání a zobrazení celé aplikace probíhá tak, že Reactu dáme povel, do kterého HTML elementu chceme vložit naší aplikaci neboli nejvyšší komponentu ve stromě. React ji z komponent poskládá (tzn. vytvoří stromovou strukturu HTML elementů). Při změně stavu aplikace (např. změnu barvy textu) knihovna sama zařídí, aby se změna v elementu projevila.

React pracuje s virtuálním DOMem, čiž znamená že nevytváří přímo daný HTML element v DOMu prohlížeče, ale má svůj vlastní. Ten při každé změně porovnává, zda se opravdu změnil a potom teprve udělá změnu v reálném DOMu. Důvodem je optimalizace [2].

Komponenty

Jak již bylo zmíněno, komponenty spolu komunikují pomocí properties. Properties fungují na podobném principu jako klasické atributy u HTML elementů a to tak, že se předávají jako atributy při zápisu komponenty. Properties mohou být jakéhokoliv typu včetně funkcí. Funkce se mohou využívat jako callback, který se zavolá po nějaké interakci uvnitř komponenty. Příkladem může být třeba kliknutím na tlačítko, kdy komponenta s tlačítkem zavolá funkci rodičovské komponenty pro obsloužení dané události. Součástí zavolání funkce mohou být i parametry.

Kromě properties může komponenta obsahovat i states (stavy). Ty vidí pouze ona sama a sama je taky může upravovat. States a properties udávají stav dané komponenty. Kdykoliv se jedna z hodnot změní, React opětovně vyrenderuje komponentu.

Následující kód ukazuje zápis jednoduché komponenty v jazyce JSX. Komponenta přijímá property s názvem `name`, který vykreslí v elementu `<div>` text *Hello* a hodnou `name`. Poslední řádek kódu vezme danou komponentu, předá ji vstupní parametr a nechá ji vykreslit do elementu `<body>`.

```
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}

ReactDOM.render(<HelloMessage name="John" />, document.body);
```

V Reactu existují 3 zápisy komponent. Každý má své výhody i nevýhody.

- Nejprostším zápisem jsou pure function. Komponenta se zapisuje pouze jako funkce, která na základě properties vyrenderuje svůj obsah. Příkladem může být komponenta, která příjme jako properties číslo a vrátí ho neformátované.
- Dalším druhem komponent jsou takové, které se vytváří jako klasické třídy poděděním od jiné třídy. Tyto komponenty dokáží kromě properties pracovat i se states viz. níže. Tento zápis vyžaduje standard JavaScriptu alespoň ECMAScript 6.
- Posledním zápisem jsou komponenty, které se vytváří tak, že se předpis komponenty vloží jako parametr do patřičné funkce a ta komponentu vrátí. Mají podobné vlastnosti jako výše uvedené komponenty vytvořené poděděním.

Komponenty prochází určitým životním cyklem, který začíná vznikem komponenty (přidáním do stromu), pokračuje změnami stavů a jejích ukončením (odebráním ze stromu). Někdy je potřeba vyvolat akce právě v jednom ze stádií života komponenty. React to řeší tak, že pro tyto účely má vyhrazené speciální metody, které si uživatel může sám implementovat. Tyto metody jsou automaticky zavolány v průběhu života a lze jimi řešit různé situace. Následuje seznam těchto metod [17][9].

- `constuctor()` je zavolán před přidáním do stromu, lze v něm nastavit states.
- `componentWillMount()` je zavolán těsně před přidáním do stromu, lze v něm nastavit states.
- `componentDidMount()` je zavolán těsně po přidání do stromu. Dobré místo pro vytvoření dotazu na server pro stažení dat.

- `componentWillReceiveProps()` invokuje se při obdržení nových props ještě před vyrenderováním. Používá se při změně states, které jsou závisle na props.
- `shouldComponentUpdate()` slouží pro optimalizaci.
- `componentWillUpdate()` invokuje se podobně jako *componentWillReceiveProps* až na to, že reaguje i na změnu states.
- `componentDidUpdate()` invokuje se po renderingu způsobeném změnami props, či states.
- `componentWillUnmount()` invokuje se před odebráním ze stromu. Může v ní zrušit žádosti, které se v průběhu života vytvořili.

Optimalizace

Byla už zmínka o tom, jak funguje virtual DOM a renderování komponent. V případě že v jedné z komponent dojde k nějaké změně states nebo properties, je potřeba ji znovu vyrendrovat. V tom případě je ale potřeba vyrendrovat i ty, které jsou ve stromě pod danou komponentou, protože v nich mohlo dojít také ke změně. Když dojde ke změně v té nejvyšší, přerenderují se i všechny ostatní. To může být výpočetně dost náročně a v mnoha případech i zbytečné, neboť ne všechny ostatní komponenty se mohly změnit. React nabízí možnost optimalizace. Do komponenty lze implementovat metodu `shouldComponentUpdate(nextProps, nextState)`, která se zavolá před vyrenderováním. Dle návratové hodnoty ověří, zda je potřeba komponentu renderovat znovu. Příkladem takové implementace může být následující kód. Ukázka popisuje kontrolu, zda nedošlo ke změně property `name`. V kladném případě se provede.

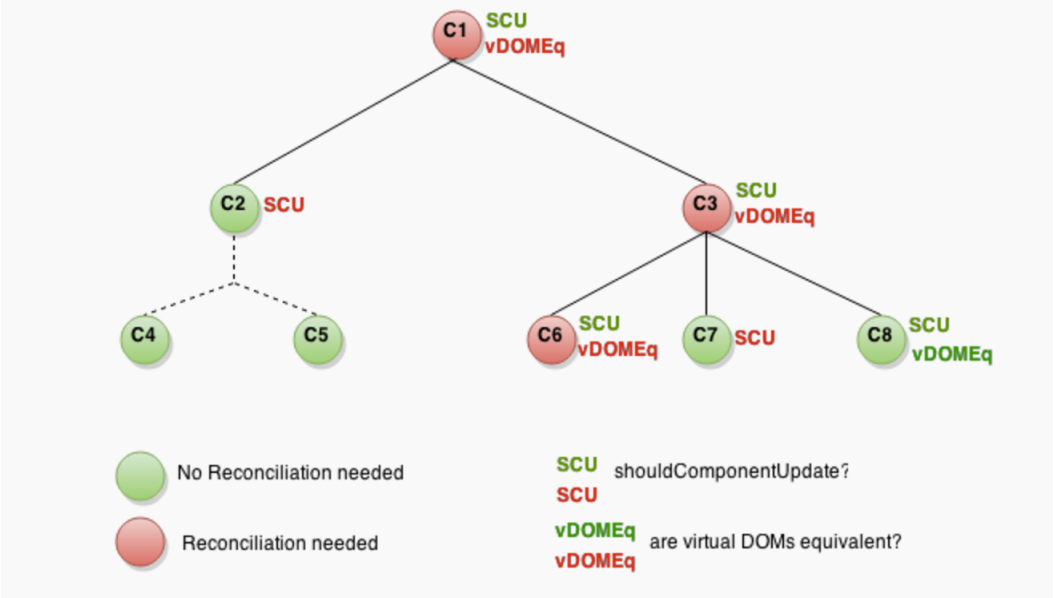
```
shouldComponentUpdate(nextProps, nextState) {
   return nextProps.name !== this.props.name
}
```

Na obrázku 2.9 je znázorněné, jak se ve stromu provádí renderování s využitím optimalize. V komponentě C1 došlo ke změně, zkontrolovalo se, zda je potřeba překreslit C2 a C3. Komponenta C3 vrátila ve své metodě `shouldComponentUpdate` hodnotu `true`. To zapříčinilo provedení renderingu, který vyvolal kontrolu u C6, C7 a C8. Jediná komponenta C6 se změnila, a proto se provedl její rendering [9].

React router

V kapitole 2.1 bylo napsáno, že SPA již nejsou prolinkované, tak jak tomu je u klasického konceptu. U něj platilo, že stav aplikace byl závislý převážně na aktuálním URL. S příchodem SPA se už ale jedná pouze o jednu aplikaci na jedné URL. React Router je plugin, který do Reactu přináší možnost vázat stav na URL. Při změně URL v takové aplikaci se neprovede nový požadavek na server, ale akce je odchycena JavaScriptem. Ten ji zpracuje a změní stav. I přes to se v historii prohlížeče udržuje stav, že došlo ke změně, takže se lze vrátit zpět pomocí nativního tlačítka prohlížeče.

React router zastřešuje kompletní práci s routrováním. Využívá k tomu speciálních komponent jako je například `<Link to="/home/»`, která je náhradou za klasický element `<a>`.



Obrázek 2.9: Renderování stromu, převzato z [9].

Routrování lze staticky nastavit či dynamicky upravovat. Funguje na způsobu měnění komponent pro různé URL. Tyto komponenty do sebe mohou být zanořené. Příkladem nastavení routrování pro aplikaci může být následující:

```
render((
  <Router history={browserHistory}>
    <Route path="/" component={App}>
      <Route path="about" component={About}/>
      <Route path="users" component={Users}>
        <Route path="/user/:userId" component={User}/>
      </Route>
      <Route path="*" component={NoMatch}/>
    </Route>
  </Router>
), document.getElementById('root'))
```

Parametr `path` udává k jaké URL se přiřadí daná komponenta. V ukázce lze vidět zanoření cest. Pokud URL nebude odpovídat žádnému pravidlu, zvolí se komponenta `NoMatch`. V URL mohou být i parametry potřebné pro UI, které jsou komponentě předané pomocí props [5].

Redux

Každá aplikace má nějaký svůj stav, který může být daný hodnotami proměnných, DOMem, URL a dalšími. Vizuální a funkční podoba by měla vycházet z daného stavu. Pokud by byl stav aplikace roztříštěn, mohou nastávat nepředvídatelné stavy. Proto je vhodné mít stav více pod kontrolou a udržovat jej. K tomu slouží knihovna Redux, jenž je stavovým kontejnerem pro jazyk JavaScript. V aplikaci udržuje jediný zdroj pravdy. Skládá se ze 3 části a to **store**, **akce** a **reducer**.

Ve storu jsou uložena všechny data aplikace. Pokud chce aplikace získat data, invokuje metodu `getState()`, která je vrací. Akce vyvolává aplikace, pokud chce udělat změnu stavu. Akce má povinný parametr `type`, který musí být pro každý typ akce unikátní. Store obsahuje metodu `dispatch`, která přijímá akci a aplikuje ji. Poslední částí je reducer, ten mění stav store na základě aktuálního stavu a zavolané akce.

Stav aplikace je potom dán výchozím stavem a zavolanými akcemi. Redux funguje na principu funkcionálního programování. Reducer je pouze funkce, která nemá žádný vnitřní stav. Na základě akce a stavu vrátí nový stav. Akce je pouhý objekt reprezentovaný parametry. Díky tomu se aplikace dobře ladí, neboť vždy máme přesný pohled na stav aplikace. Ladící nástroje umožňují stavy aplikace vracet zpět.

Redux se dá snadno zkombinovat s Reactem a tvoří tak návrhový vzor. Dá se kombinovat s React Router, kde URI rozšiřuje stav. Pro asynchronní události lze využít Middlewary, které toho ovšem nabízí víc[1].

Immutable

V předchozí části byl popsaný reducer, jakožto funkce, která příjme stav a akci. Následně vrátí nový upravený stav. Tato modifikace se děje takovým způsobem, že se celý stav vytvoří nový a vrátí upravený. Důvodem je, že reducer nemůže mít žádné vedlejší účinky. Při kopírování stavu se často mění jen jedna z hodnot a zápis takové úpravy v čistém JavaScriptu může být nepřehledný a zdlouhavý (je to dáno tím jak JavaScript pracuje s objekty). Tento problém řeší návrhový vzor immutable, který popisuje, že nelze měnit nastavené hodnoty. Pokud chceme upravit objekt, musí vzniknout nový. Facebook stojí za vznikem knihovny immutable.js, která funguje dle návrhového vzoru. Při žádosti o změnu objektu, dojde k jeho naklonování a vrácení nového upraveného. Je tedy zaručeno, že k úpravě stávajícího nikdy nedojde. Knihovna nabízí pokročilou práci s těmito typy objektů, například:

- **List** klasický seznam,
- **Map** neseřazené pole dvojic klíč hodnota,
- **Stack** zásobník,
- **Range** seřazený ohraničený seznam číselných hodnot s definovaným rozestupem,
- **Repeat** seznam opakujících se hodnot [8].

React Bootstrap

Knihovna React Bootstrap slouží pro snadnější integraci již zmíněného frameworku Twitter Bootstrap do aplikace napsané v Reactu. Obsahuje většinu částí frameworku. Práce s prvky probíhá tak, že je uživatel přidává jako komponenty, přičemž pomocí properties ovlivňuje jejich vlastnosti. Knihovnu lze nainstalovat klasickým způsobem pomocí *npm*. Navíc je potřebné nahrát bootstrap styly. Následující kód ukazuje, jak vypadá zápis komponenty *Button* v JSX oproti klasickému HTML[10].

```
<button type="button" class="btn btn-success btn-sm">
  HTML button
</button>
```

[10] Dokumentace https://react-bootstrap.github.io

```
<Button bsStyle="success" bsSize="small">
   React button
</Button>
```

Kapitola 3

Existující aplikace

V následující kapitole je popsáno, jak problém vyřizování reklamací v internetových obchodech řeší některé e-shopy. V první části 3.1 je ukázáno, jak lze vytvořit reklamační požadavek u třech českých obchodů. Ve druhé části 3.2 je výčet několika pluginů, které lze nainstalovat do internetových obchodů, založených na některém z e-shopových platforem. Tyto moduly rozšiřují možnosti práce s reklamačními požadavky.

3.1 Vlastní systémy e-shopů

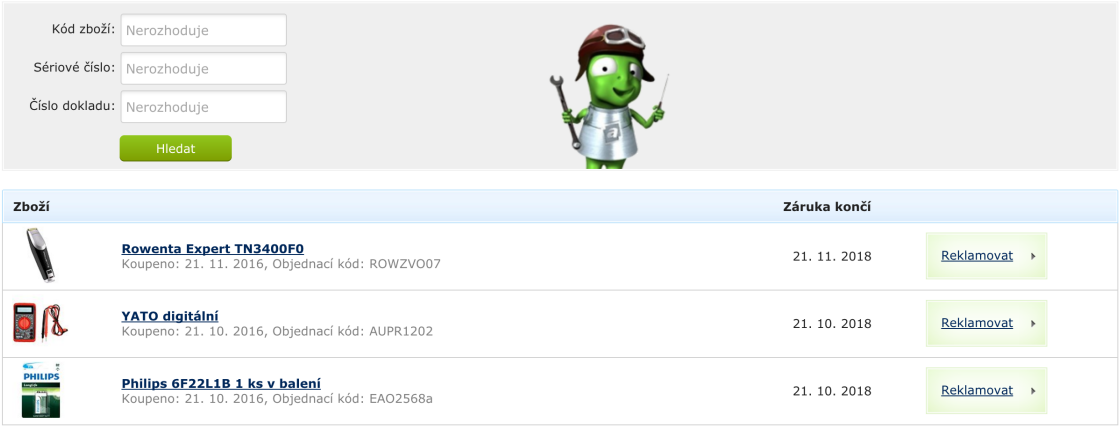
V této sekci jsou rozebrané způsoby, jakým lze reklamovat zboží u vybraných českých internetových obchodů. U všech následně zmíněných e-shopu je možné zboží reklamovat přímo na pobočce nebo jej zaslat přepravní společností. Přičemž obchody umožňují sledovat stav reklamace na svých stránkách. Jsou zde popsány pouze procesy vytvoření reklamací.

Alza

Internetový obchod Alza nabízí dva způsoby, jakým lze reklamovat zboží pomocí přepravní společnosti. Prvním způsobem je možnost zaslat zboží společně s dokladem o koupi a průvodním dopisem, který popisuje konkrétní vady a obsahuje kontaktní údaje. Druhou možností je vytvořit si online RMA číslo, které se poté přiloží ke zboží a je možné pomocí něj sledovat stav reklamace.

Vytvoření RMA čísla je možné pouze pro přihlášeného uživatele, jenž provedl objednávku. Obchod má pro reklamace speciální záložku, na kterou se lze dostat v sekci uživatelského účtu. V horní části stránky je filtr, pro vyhledání zboží na základě parametrů. Pod ním se nachází seznam objednaných produktů. U každého produktu je informace o tom, kdy končí záruka a základní informace o prodeji viz. obrázek 3.1. Na pravé straně se nachází tlačítko *Reklamovat*. Po jeho kliknutí se otevře nové dialogové okno, ve kterém se nachází 3 kroky. V prvním kroku si uživatel zvolí preferovaný způsob vyřízení a popíše závadu. V následujícím kroku si zvolí způsob dodání a v posledním kroku si určí způsob vyzvednutí. Zároveň může přiložit soubor. Poté se zobrazí RMA kód. Vytvořené reklamace se poté nachází v podsekci *Aktivní reklamace*. V této části je možné sledovat stav reklamací a nebo položit obchodu dotaz k dané reklamaci[1].

[1]Stránka obchodu Alza www.alza.cz



Obrázek 3.1: Alza reklamace

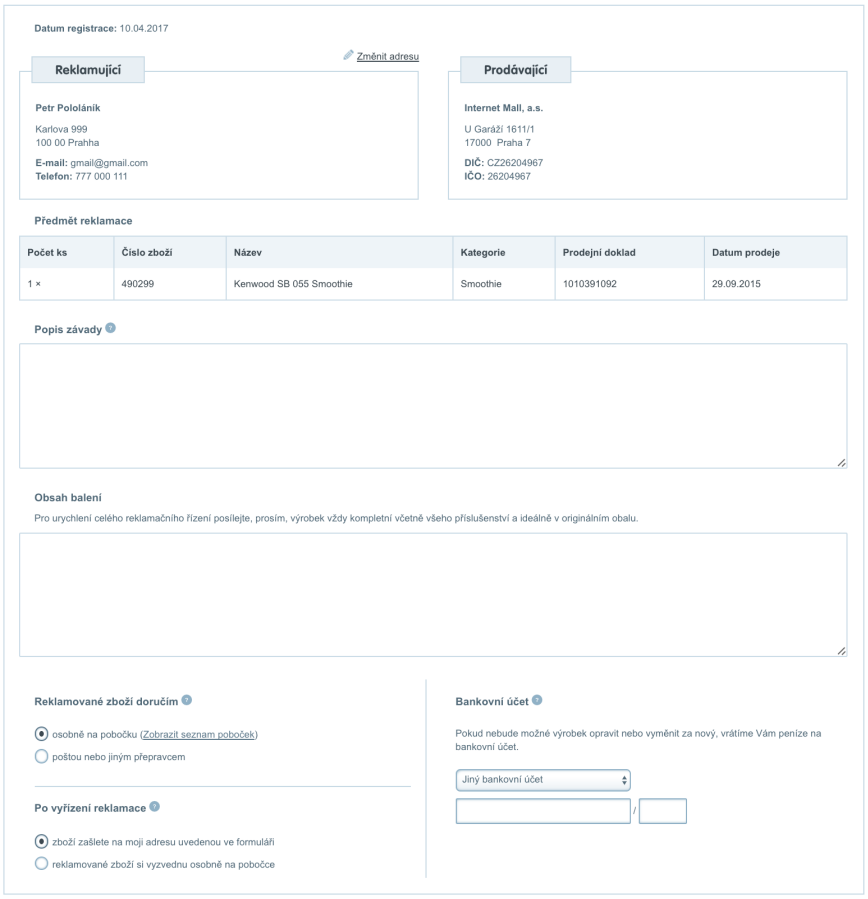
Mall

V obchodu Mall je stejně jako v Alza.cz možné zboží přímo zaslat nebo využít online registraci reklamace. Pro druhý způsob je nutné se do obchodu přihlásit. Poté vstoupit do sekce *Moje objednávky*, ve kterém je seznam všech objednávek s popisem. U každé objednávky se nachází odkaz s názvem *reklamovat*, po jehož kliknutí se stránka přesměruje do části s názvem *Chystáte se reklamovat*. V této části obchod nejprve vyzývá uživatele, aby reklamoval zboží přímo v autorizovaném servise a urychlil tím celý proces. Zároveň informuje o tom, že pokud je zboží nadrozměrné, může dorazit servisní technik přímo k výrobku. Aby uživatel mohl pokračovat, musí stisknout zatrhávací box s popisem *Reklamaci chci uplatnit v MALL.CZ* a následně kliknout na tlačítko pokračovat. Tím se mu zobrazí Registrační formulář viz. obrázek 3.2. V tomto formuláři uživatel vyplní vše potřebné, jako je popis závady, kontaktní údaje, způsob zaslání, způsob doručení a jiné. Následně klikne na tlačítko *Pokračovat*, vše si zkontroluje a ještě jednou zadané údaje potvrdí kliknutím na tlačítko *Pokračovat*. Poté v sekci *Moje reklamace* najde danou reklamaci, kde je možné sledovat její stav. Při předání či odeslání zboží je nutné přiložit vytištěný reklamační protokol[2].

CZC.cz

Obchod s elektronikou CZC.cz podobným způsobem jako předchozí obchody dává možnost reklamované zboží zaslat bez předchozího vytvoření žádosti. Avšak přikládá k tomuto způsobu vzorový formulář, který lze vytisknout, vyplnit a přiložit ke zboží. Vytvoření registrace online je možné v sekci *Moje nákupy*, kde je nutné zboží vyhledat v podsekci *Objednávky*, jenž je přehledem všech uskutečněných objednávek. Je zde možné vyhledávat pomocí různých kódů a názvu zboží. U každé položky se nachází tlačítko *reklamovat*, jehož kliknutím se uživatel dostane do reklamačního formuláře. Do tohoto formuláře se lze dostat i přes podsekci *Reklamace*, kde je tlačítko *Přidat novou reklamaci*, jenž přesměruje do podobného seznamu jako je seznam objednávek. V reklamačním formuláři viz. obrázek 3.3 je nutné vyplnit popis závady. Uživatel si dále vybere způsob vyřízení, způsob dopravy a dodací adresu. Dodací adresu u tohoto obchodu nelze zvolit libovolnou. Vytvořený požadavek se poté zobrazí v podsekci *Reklamace*, v níž lze sledovat jeho stav[3].

[2]Stránka obchodu Mall www.mall.cz
[3]Stránka obchodu CZC.cz www.czc.cz



Obrázek 3.2: Mall reklamační formulář

Obrázek 3.3: CZC.cz reklamační formulář

3.2 Doplňky pro e-shopy

Všechna níže zmíněná existující řešení jsou stejná v tom, že neumožňují použití pro libovolný obchod. Nabízí se skrze plugin pro nějakou z platforem pro e-shopy. Následuje seznam a stručný popis několika takovýchto pluginů.

Returnly obsahuje práci s dopravcem, kterého je možné přímo objednat. Při vyřizování zákazník rovnou vidí, kolik peněz mu obchod vrátí. Podporuje notifikaci o změnách pomocí e-mailu. Pro zákazníky je plně responzivní, vše lze jednoduše vyřídit na mobilním telefonu. Tato služba se platí paušálně každý měsíc. Cena závisí na počtu žádostí a začíná na 9 $[4].

Returns Manager je modul, který pro platformu *Shopify* vytvořila společnost *Bold*. Aplikace obsahuje základní věci, avšak neumožňuje vrátit více položek v jedné žádosti. Cena služby je fixní částka, která se platí každý měsíc a to 19,99 $[5].

Returns and Warranty Requests je produkt, který si sám pro svoji platformu vytvořil *woocommerce*. Cena se liší dle počtu zakoupených licencí, přičemž licence nejsou časově omezené. Pro jeden e-shop aplikace stojí 79 $ a je v ní zahrnuta i roční podpora[6].

Ticket system je na tom podobně jako **Returns and Warranty Requests**, je vytvořen vývojáři z *Shopware* pro jejich vlastní platformu. Měsíční licence stojí 30 €[7].

Automated RMAs and Returns je plugin pro *Shopify*. Cena činí 19,95$ měsíčně[8].

Jako možné řešení zpracování reklamací by se daly chápat i helpdesky. Ovšem tyto nástroje nejsou zaměřené na daný úkol. Vyřídit reklamaci s nimi možné je, ale nejsou k tomu dostupné takové nástroje jako nabízí specializované pluginy či aplikace.

[4] Returnly http://www.returlny.com/
[5] Returns Manager https://apps.shopify.com/returns-manager
[6] Returns and Warranty Requests https://woocommerce.com/products/warranty-requests/
[7] Ticket system http://store.shopware.com/
[8] Automated RMAs and Returns http://www.automatedrmas.com/

Kapitola 4

Zhodnocení současného stavu a plán práce

V této kapitole je zhodnoceno, jakým způsobem řeší reklamaci běžné e-shopy (podkapitola 4.1) a jak lze vytvářet klientskou část webové aplikace v podkapitole 4.2. Následně jsou popsané požadavky na výsledný software zpracovávaný v rámci této práce 4.2. V poslední části 4.3 jsou stručně popsané role uživatelů, kteří budou v aplikaci figurovat.

Do spolupráce s implementací aplikace se připojila firma Awexa s.r.o., jejíž náplň je navržení a vytvoření serverové části aplikace. Návrh komunikace mezi klientem a serverem včetně toho, co bude server nabízet za služby a jak budou fungovat, je řešené v rámci této diplomové práce. Stejně tak i celá klientská aplikace.

4.1 Zhodnocení současného stavu

V současné době e-shopy postavené na většině rozšířených platformách mají možnost si nainstalovat doplněk do systému, který obchod rozšiřuje o vytváření a správu reklamací. Doplňků je mnoho, přičemž jejich možnosti se liší dle platforem. Obchody postavené na vlastním systému si reklamace řeší většinou vlastním způsobem.

Pro dynamickou webovou aplikaci je vhodný koncept Single Page Application, kdy je velká část aplikační logiky přenesena na klienta. Pro rychlou komunikaci bez nutnosti stálého navazování spojení mezi klientem a serverem jsou vhodné WebSockets. Nástroje pro vytvoření grafického uživatelského prostředí proces tvorby aplikace zjednodušují a zrychlují. Díky frameworkům, jako je například Twitter Bootstrap, je možné rychle vytvořit prototyp aplikace a zaměřit se více na její funkčnost než na grafický styl, ten lze odložit na později. Preprocesory a postprocesory CSS přidávají možnost snadněji a přehledněji zapisovat kaskádové styly. Knihovna React umožňuje vytvořit dynamické komponenty a pomocí dalších knihoven dostupných přes npm vytvořit plnohodnotnou aplikaci v JavaScriptu dostupnou přes prohlížeč. Pomocí NodeJS, a například na něm postaveném nástroji Webpack, se může struktura aplikace rozčlenit do různých částí a ty nakonec spojit. Díky tomu jsou zdrojové kódy při vývoji přehlednější a lze bezpečněji vytvořit komplexnější aplikaci. Pro aplikaci určenou k reklamování zboží jsou tyto prostředky velice vhodné, neboť se jedná o komplexnější systém, který by měl mít rychlou odezvu a nemusel bz být závislý na obnovování stránky.

4.2 Požadavky na výsledný software

Výsledná aplikace musí umožnit jakémukoliv e-shopu vyřizovat reklamace a to bez zásahu do stávajícího systému, na kterém je e-shop postaven. Vyřizováním reklamací je myšleno, že zákazník musí mít možnost podat přes aplikaci žádost o reklamaci a celý její proces sledovat online.

Obchod musí mít možnost si upravovat chování aplikace dle svých potřeb, konkrétně si vytvářet vlastní typy reklamačních žádostí. Takovou například může být vrácení zboží, výměna za jinou velikost a libovolné jiné dle potřeby obchodu. Musí být umožněno, aby si obchod pro každý takový typ mohl vytvořit vlastní formulář, který zákazník bude při vytváření žádosti vyplňovat.

Během procesu reklamace se mění její status (vytvořeno, zboží odesláno, spravuje se, a jiné...). Obchod musí mít možnost si vytvořit libovolné statusy a zaměstnanci je poté přiřazovat k žádostem. Zaměstnanec musí mít možnost k žádosti, kterou spravuje, přidat dodatečné interní informace, jenž jsou zákazníkovi skryté. Je nutné, aby se reklamace mohla přiřadit k zaměstnanci, který ji má na starost. Zaměstnanci musí mít možnost si mezi reklamacemi v obchodu filtrovat, vyhledávat a řadit.

Zákazník musí mít možnost ke svým reklamacím nahrát obrázky a jiné soubory. Obrázky musí být možné prohlížet online v aplikaci. Je nutné, aby zákazník mohl komunikovat se zaměstnancem a zároveň měl přehled o změnách, případně být i nějakým způsobem notifikován. Pro komunikaci mezi zákazníkem a zaměstnancem musí být dostupný chat.

Aplikace musí být přístupná přes webový prohlížeč. Zákazníci i zaměstnanci obchodu musí mít možnost aplikaci ovládat plnohodnotně i přes tablet či mobilní telefon. Design tedy musí být responsivní. Změny dat v aplikaci by měli být vidět bez manuálního obnovení stránky v prohlížeči a to ihned jakmile jsou dostupné.

4.3 Požadavky na role

V aplikaci by měly existovat 3 role uživatelů. Uživatelé, kteří budou vytvářet požadavky na reklamaci zboží, by se měli nazývat **zákazníci**. Zákazníky by měli obsluhovat **zaměstnanci** daných obchodů. Posledním typem uživatelů by měli být **administrátoři**. Ti musí mít možnost do aplikace přidávat nové obchody. Administrátoři by měli zároveň mít možnost do svých obchodů přidávat své zaměstnance. Všichni uživatelé, kteří budou typu zaměstnanec, by tedy měli být přidáni jedním z administrátorů. Zároveň pokud administrátor nebude chtít zaměstnance u svého obchodu, musí mít možnost vystupovat sám jako zaměstnanec. Měl by převzít roli zaměstnance svého obchodu. Zákazníci a administrátoři by se do aplikace měli registrovat sami bez omezení.

Aby se uživatel registrovaný jako administrátor mohl stát i zákazníkem, měl by mít možnost přepnout se do role zákazníka. Opačný případ musí být také možný, tedy zákazník musí mít možnost se přepnout na administrátora a vytvořit si vlastní obchod. To umožní jednotnou registraci. Shrnutím lze říct, že uživatel přidaný jako zaměstnanec bude vždy pouze zaměstnancem. Registrovaní uživatelé by se měli přepínat mezi rolemi zákazník a administrátor, v případě že vytvoří obchod, tak i zaměstnancem.

Následující seznamy popisují jaké možnosti musí jednotlivé role v aplikaci mít a co mohou provádět:

- Zákazník
  - vytváření požadavků

  - přidávání a zobrazení souborů u požadavku
  - vyplňování zákaznického formuláře u požadavku
  - chat se zaměstnanci
  - přehled všech požadavků
- Zaměstnanec
  - změna statusu požadavku
  - zobrazení zákaznického formuláře
  - úprava zaměstnaneckého formuláře
  - přidávání a zobrazení souborů u požadavků
  - chat se zákazníkem
  - přehled všech požadavků obchodů u nichž je jako zaměstnanec
- Administrátor
  - vytváření a správa obchodů
  - vytváření a správa typů požadavků
  - vytváření a správa typů statusů
  - vytváření a správa zaměstnanců
  - přidělení zaměstnance k obchodu
  - úprava struktury interních formulářů (u požadavku)

Kapitola 5

Implementace

Tato kapitola se zabývá návrhem, implementací a testováním výsledné aplikace. Ze začátku jsou navrženy a popsány některé základní stránky aplikace 5.1. Součástí každého návrhu je i grafické znázornění. Na to navazuje podkapitola 5.2, ve které je popsán základní koncept aplikace a komunikace mezi klientem a serverem včetně ukázek zpráv a dat. Poté jsou popsané některé z prvků v aplikaci například formuláře nebo překlady stránek. Ke konci podkapitoly je ukázka stránky detailu reklamace. Ke konci kapitoly je demonstrace použití výsledné aplikace na ukázkovém obchodu 5.3. V poslední podkapitole 5.4 jsou popsané a navržené některé z testů aplikace pro produkční nasazení.

5.1 Návrh a popis obrazovek

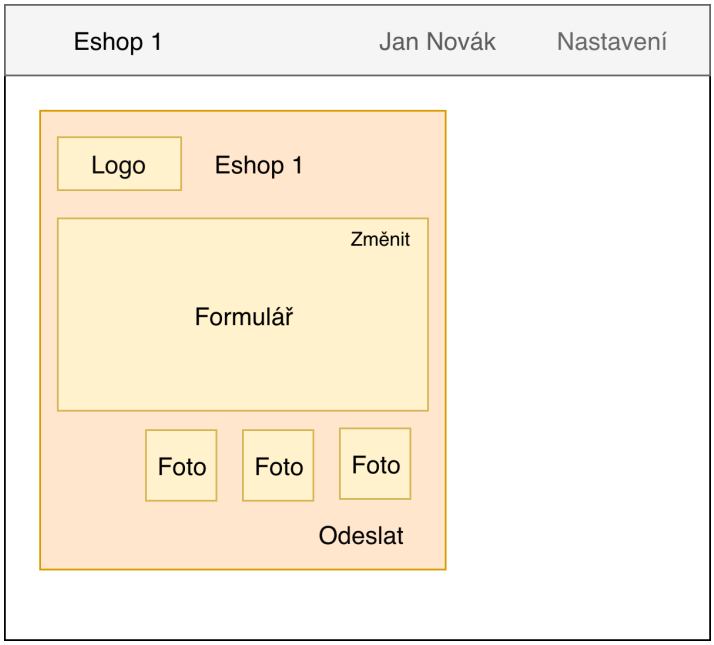
V následující sekci je popsán návrh většiny obrazovek. Každý typ uživatele se bude přihlašovat stejnou přihlašovací obrazovkou. Na této obrazovce bude pouze základní popis aplikace a přihlašovací formulář s možností přepnout jej na registraci pro nové uživatele. Při vyplňování registračního formuláře bude probíhat kontrola správně vyplněných dat ještě před odesláním na server.

Zákazník

Návrh obrazovky pro vytvoření nové reklamace zákazníkem je znázorněn na obrázku 5.1. Do této obrazovky se zákazník dostane přes odkaz, který mu poskytne obchod (to jakým způsobem obchod odkaz vytvoří, bude popsáno v sekci návrhu obrazovky administrátora).

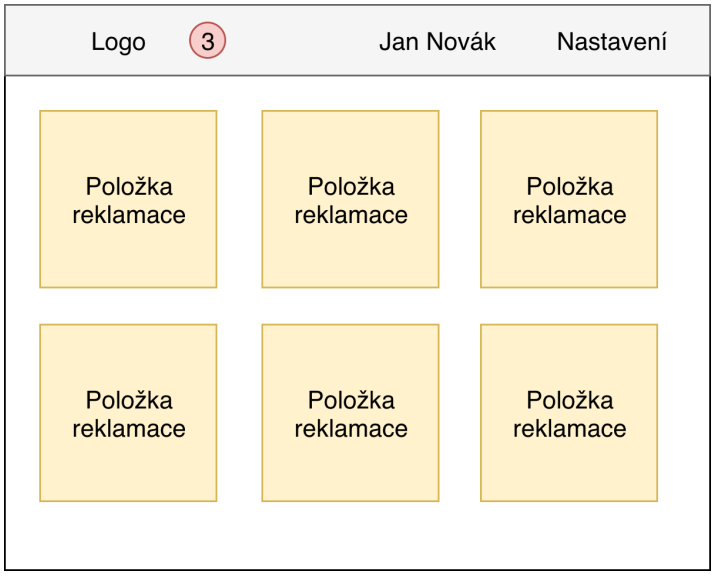
Tato obrazovka bude jak pro přihlášené, tak i anonymní uživatele. U anonymních uživatelů nebude zobrazeno jméno uživatele a možnost přejít do nastavení. V levé horní části bude zobrazen název obchodu, u kterého se reklamuje. Pod ním bude box s reklamačním formulářem. Na vrchu formuláře bude logo obchodu (pokud je zadané) a název. Pokud má obchod více typů reklamačních formulářů, na pravé straně bude tlačítko *změnit*. Po kliknutí na toto tlačítko se zobrazí nabídka pro změnu formuláře. Součástí bude i možnost přidat soubor, přičemž u některých přidaných souborů bude i jejich náhled nebo ikony znázorňující typ souboru. Posledním prvkem je tlačítko pro odeslání formuláře. Pokud bude uživatel anonymní, po kliknutí na toto tlačítko se uživateli zobrazí okno s výzvou, aby se přihlásil či registroval.

Na obrázku 5.2 je znázorněna vstupní obrazovka přihlášeného zákazníka. V horní části stránky bude kruh s číslem uvnitř, který bude sloužit pro notifikaci nepřečtených zpráv. V případě, že uživatel bude mít aspoň jednu nepřečtenou zprávu, bude kruh červený a číslo



Obrázek 5.1: Návrh obrazovky zákazníka - nový požadavek.

bude počet nepřečtených zpráv. Pokud uživatel bude mít všechny zprávy přečtené, kruh bude méně výrazný a bude ukazovat celkový počet zpráv.

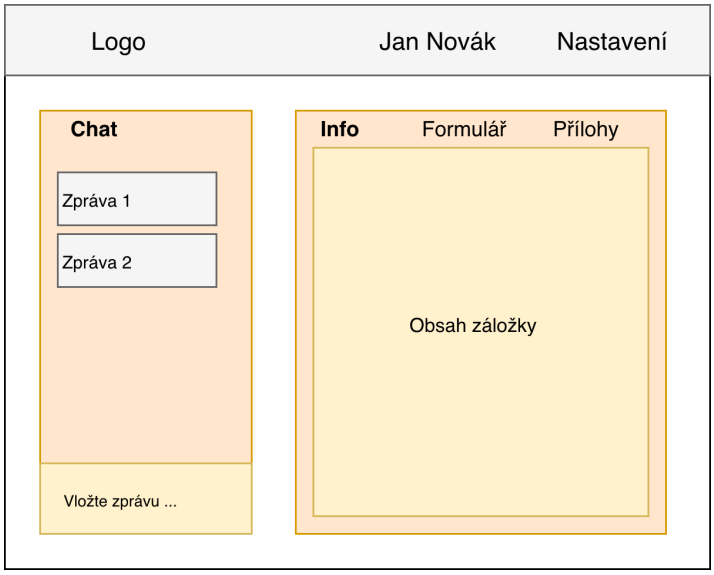


Obrázek 5.2: Návrh obrazovky zákazníka - seznam požadavků.

Na této obrazovce zákazník uvidí seznam svých reklamací seřazený podle data vytvoření od nejnovější reklamace po nejstarší. Každá položka reklamace bude obsahovat základní informace a to název obchodu, datum vytvoření, typ žádosti a stav. Kliknutím na jednu z položek se zákazník dostane do detailu dané reklamace.

Návrh detailu reklamace je znázorněn obrázkem 5.3. Obrazovka je rozdělena do dvou částí. V levé části je chat pro komunikaci se zaměstnanci obchodu. Zprávy jsou seřazené ze

shora od nejstarší po nejnovější. Pod zprávami je prvek pro vložení nové zprávy. V pravé části se uživatel pomocí tabulátorů bude moci přepínat mezi sekcemi Info, Zákaznický formulář a Přílohy. V sekci **info** uvidí zákazník informace o dané reklamaci - statusy, datum vytvoření, jméno zaměstnance, typ formuláře a další. Zároveň v této záložce může být zobrazena speciální nabídka od obchodu. Příkladem takové nabídky může být například kód voucheru pro slevu na další nákup. Záložka **Zákaznický formulář** zobrazí formulář, který zákazník vyplnil při vytváření žádosti. Formulář bude moci dodatečně upravit. Pokud obchod bude umožňovat více typů žádostí, zákazníkovi bude umožněno se dodatečně přepnout na jiný. Poslední záložka s názvem **Přílohy** bude zobrazovat seznam nahraných souborů k reklamaci a to včetně souborů, které nahrál zaměstnanec. U některých souborů bude dostupný jejich náhled a po kliknutí na ně i zobrazení.



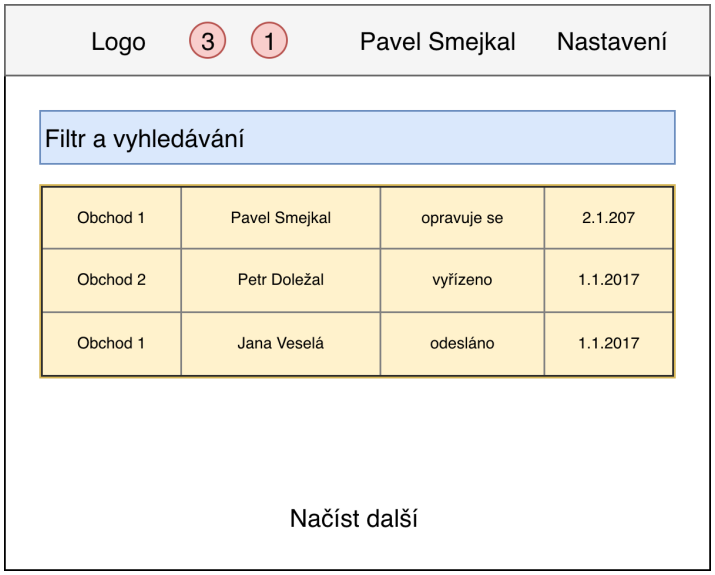
Obrázek 5.3: Návrh obrazovky zákazníka - seznam požadavků.

Zaměstnanec

Po přihlášení zaměstnanec uvidí seznam všech reklamací, pro jejichž obchody je přiřazen jako pracovník. Tento návrh je znázorněn na obrázku 5.4.

Požadavky uvidí seřazené pod sebou v tabulce. U každého požadavku budou základní informace o reklamaci a počtu zpráv v chatu. Pokud daný zaměstnanec bude mít nepřečtenou zprávu u reklamace, kterou řeší, bude o tom upozorněn pomocí notifikace na daném řádku. Reklamací může mít mnoho a z toho důvodu se nezobrazí všechny, ale budou se postupně načítat tak, jak se bude uživatel postupně pohybovat na obrazovce směrem dolů. Pokud by na danou událost prohlížeč špatně reagoval, bude možnost vyvolat načtení i kliknutím na tlačítko pod tabulkou. V horní části nad požadavky bude filtr a možnost řazení. Filtr bude umožňovat i textové vyhledávání podle ID požadavku. Zaměstnanec bude moci filtrovat dle obchodů, pracovníků, statusů a typů požadavků.

Po kliknutí na položku reklamace se uživatel dostane na obrazovku detailu reklamace. Tato obrazovka bude podobná jako obrazovka detailu reklamace u zákazníka viz. obrázek 5.3. Rozdíl bude v pravé části, konkrétně v počtu a obsahu záložek. Přibude záložka **Formulář zaměstnance**, která bude obsahovat interní formulář vázaný k detailu reklamace.



Obrázek 5.4: Návrh obrazovky zaměstnance - seznam požadavků.

Tento formulář bude moci upravovat kterýkoliv ze zaměstnanců. V záložce **Info** budou informace o reklamaci podobně jako u zákazníka, avšak přibude zde možnost změnit status reklamace a přiděleného zaměstnance. Také bude možné nastavit speciální obchodní akci pro danou reklamaci. Záložka **Formulář** bude totožná se zákaznickou, rozdíl bude pouze v tom, že formulář nepůjde editovat. **Přílohy** se nezmění.

Administrátor

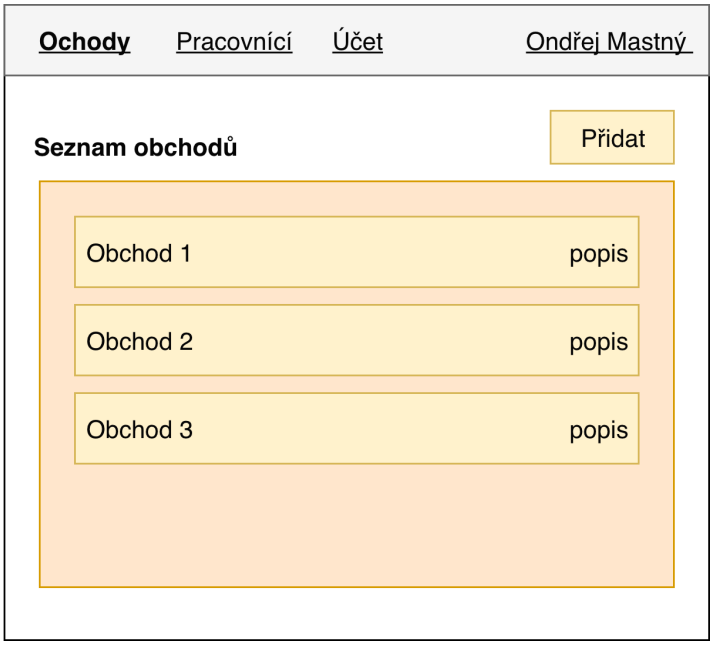
Administrátor na svých stránkách v horní části uvidí hlavičku, ve které jsou v levé části položky menu a v pravé části jeho jméno. Pomocí menu bude přecházet mezi jednotlivými sekcemi. Po kliknutí na jméno se uživateli zobrazí menu ve kterém se může přepnout na jinou roli, odhlásit se a nebo se dostat do nastavení profilu.

Po přihlášení se uživatel dostane do sekce **Obchodu**, kde uvidí seznam všech obchodů, které vytvořil. Tento seznam lze vidět na obrázku 5.5. Každý obchod je ve vlastním boxu společně s informací, zda je aktivní. Pokud je obchod neaktivní, nelze u něj provádět reklamace. Nad seznamem se nachází tlačítko *přidat*. Tímto tlačítkem se bude vytvářet nový obchod. Po kliknutí na něj se zobrazí v dialogovém okně formulář, pomocí kterého se vytvoří.

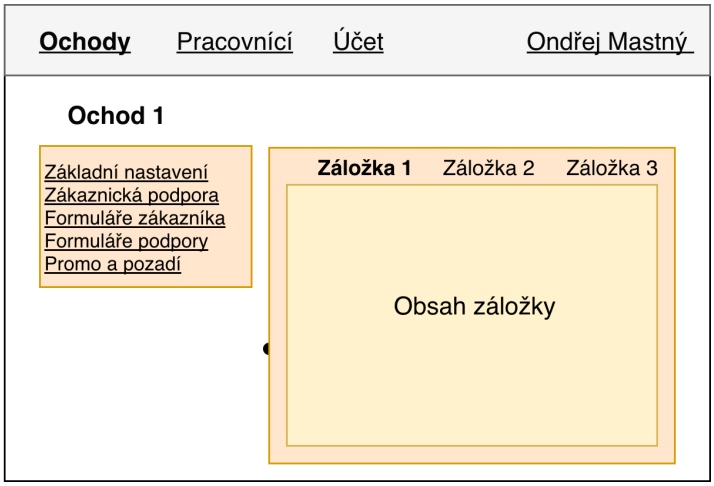
Po kliknutí na obchod se administrátorovi zobrazí detail obchodu (obrázek 5.6). V pravé části je další úroveň menu a v levé obsah podstránky. Druhá úroveň menu se váže pouze k vybranému obchodu. Součástí některých obsahů podstránek bude ještě 3. úroveň menu.

Stromovou struktura podstránek na stránce obchodu je následující:

- Základní nastavení
  - Údaje obchodu (formulář se základními údaji)
  - Barvy
  - Emailové sdělení (nastavení notifikačních emailů emailů)
  - Stavy požadavků (správa stavů reklamací),



Obrázek 5.5: Návrh obrazovky administrátora - seznam obchodů.



Obrázek 5.6: Návrh obrazovky administrátora - seznam obchodů.

  - Integrace (integrace aplikace do e-shopu pomocí odkazů)
- Zákaznická podpora
- Formuláře zákazníka
  - Seznam položek formulářů
- Formuláře podpory
- Promo a pozadí
  - Promo (správa promo akcí, které zaměstnanec může přiřadit k reklamaci),

– Pozadí (bude zobrazeno o nového reklamačního formuláře).

V záložce integrace se budou nacházet URL odkazy. Přes tyto URL bude zákazník přistupovat do části aplikace pro vytváření nových požadavků.

Na stránce **Pracovníci** bude seznam pracovníků podobný seznamu obchodu. Zde se budou vytvářet a spravovat noví pracovníci. Tito vytvoření pracovníci pak dále půjdou přiřadit k danému obchodu v jeho detailu na podstránce **Zákaznická podpora** . Na této podstránce bude stejný seznam pracovníků, jen u nich bude zaškrtávací prvek, jenž bude určovat, zda je daný zaměstnanec součástí obchodu.

5.2 Detaily implementace

Aplikace je založena na konceptu Single Pace Application. Klientská aplikace je implementovaná v jazyce JavaScript. Využívá technologii React a na něm postavených knihoven. Jako základní kostra aplikace byla použita kostra dostupná na githubu[1]. Pro grafické uživatelské prostředí je využita upravená verze knihovny Twitter Bootstrap, jejíž vzhled je modifikován do podoby Material Design. Některé komponenty této knihovny jsou v aplikaci implementovány pomocí modulu **react-bootstrap**. Styly jsou napsané pro preprocesor SASS.

Struktura aplikace

Struktura jednotlivých stránek v aplikaci se skládá z různě do sebe zanořených komponent. V aplikaci se lze pohybovat pod třemi rolemi uživatelů. Pro každou roli se obsah stránek liší, avšak některé části jsou společné. Komponenty se dělí do adresářů podle toho, zda jsou obecné (používají se na více místech) nebo určené přímo pro konkrétní stránku. Adresář pro obecné komponenty je */src/client/components/*, ostatní jsou v adresáři */src/client/-containers/*. Nastavení struktury stránek se nachází v souboru */src/client/index*.

Komunikace se serverem

Klientská aplikace se serverem komunikuje pomocí WebSockets. Zprávy se posílají ve formátu JSON. Spojení se serverem se navazuje automaticky hned po spuštění aplikace. Pokud byl již uživatel přihlášen a v prohlížeči jsou uložené přihlašovací údaje, tak se pošle zpráva pro přihlášení uživatele. V opačném případě se pošle zpráva s anonymním přihlášením. Server na každou zprávu, která mu dorazí, odpoví zprávou o doručení. Součástí každé odeslané zprávy je i unikátní identifikátor. Ten slouží k tomu, aby klient rozpoznal, na kterou zprávu se vztahuje aktuální odpověď o doručení. Následující kód popisuje strukturu zprávy, konkrétně se jedna o zprávu pro přihlášení.

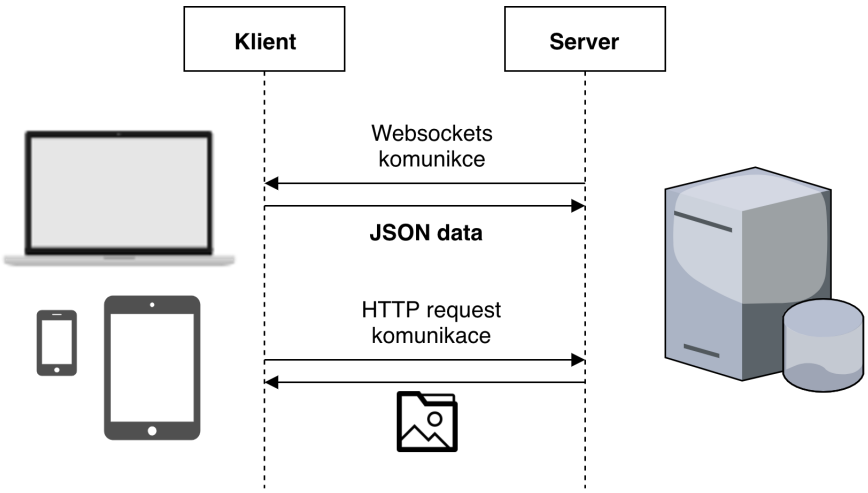
```
{
   "clientType":1,
   "clientVersion":1,
   "cmd":"get_token",
   "login":"customer@reklamace.reklamace",
   "password":"secret",
   "requestId": 1,
   "userType":1
}
```

[1] Dokumentace https://github.com/tajo/devstack

Každá zpráva má povinný parametr **cmd** určující typ zprávy a **requestId**, který slouží jako výše zmíněný identifikátor. Ostatní parametry se liší dle typu zprávy. V ukázkové přihlašovací zprávě jsou parametry **clientType** a **clientVersion** statické a nemění se. Důvod jejich existence je ten, aby se případně v budoucnu mohlo odlišit více druhů API a klientů. Parametry **login** a **password** jsou email uživatele a jeho heslo. Poslední parametr **userType** určuje o jakou roli uživatele se jedná, pokud zůstane nevyplněn, server vybere automaticky poslední zvolený. Pokud by se daný uživatel chtěl přihlásit pod jinou roli, musí vytvořit nové spojení. Pro zákazníka parametr nabývá hodnoty 1, pro zaměstnance 2 a administrátor má hodnotu 3. Jako odpověď na tuto zprávu dojde JSON, který vypadá následovně.

```
{
   "cmd": "get_token",
   "requestId": 2,
   "result": 1,
   "timestamp": 1494168791,
   "duration": 28,
   "data": {
      "aid": 281474976710740,
      "userType": 1,
      "lastUserType": 1,
      "requestTime": 1494168791387,
      "token": "DSLK46LC9DM40..."
   }
}
```

Odpověď na všechny zprávy se oproti ukázkové liší pouze ve struktuře **data**. Ostatní parametry jsou pro kontrolu komunikace. Důležitý parametr **aid** je unikátní identifikátor přihlášeného uživatele. Parametr **token** je řetězec, pomocí kterého se klient může autorizovat u serveru, pokud komunikuje mimo WebSockets. Posílá se například při žádosti o stažení nebo nahrání souborů přes klasické HTTP dotazy. Na obrázku 5.7 je schéma této komunikace.



Obrázek 5.7: Komunikace klienta se serverem.

Zmíněny princip komunikace pomocí WebSockets je implementovaný v knihovně */src/client/api*. Pro každý typ zprávy existuje vlastní metoda. Pro přihlášení je například k dispozici metoda, jenž se volá `API.getLogin(user, password, userType)`. Každá tato metoda vrací Promise objekt pro asynchronní komunikaci. Knihovna má svoji vnitřní frontu, kam jednotlivé žádosti o volání metod ukládá. Ty potom sekvenčně provádí a odebírá z fronty. Díky tomuto je zaručeno, že se požadavky nemohou předběhnout.

Data

Ihned po přihlášení klientovi dojdou aktuální data, které má právo vidět. Jakmile dojde na serveru ke změně, dojde i k jejich aktualizaci na klientovi. Klient tyto data ukládá pouze po dobu existence instance aplikace v prohlížeči. Data jsou uchována a distribuována pomocí stavového kontejneru **redux**. Všechny tyto data chodí jako množiny entit některého z následujících typů:

- avatars (fotografie uživatelů)
- file (soubory vázané na obchod)
- items (reklamace)
- status (informace o zobrazení reklamace zaměstnanci)
- messages (zprávy)
- multimedia (přílohy vázané na reklamaci)
- notifications (emailové notifikace o změně v reklamaci)
- projects (obchody)
- projects_share (zaměstnanci přiřazeni k obchodům)
- users (seznam uživatelů)

Každá entita má vlastní identifikátor a vzájemně na sebe odkazují. Takto může vypadat entita z množiny **messages**, která slouží jako položka zprávy v chatu u reklamace.

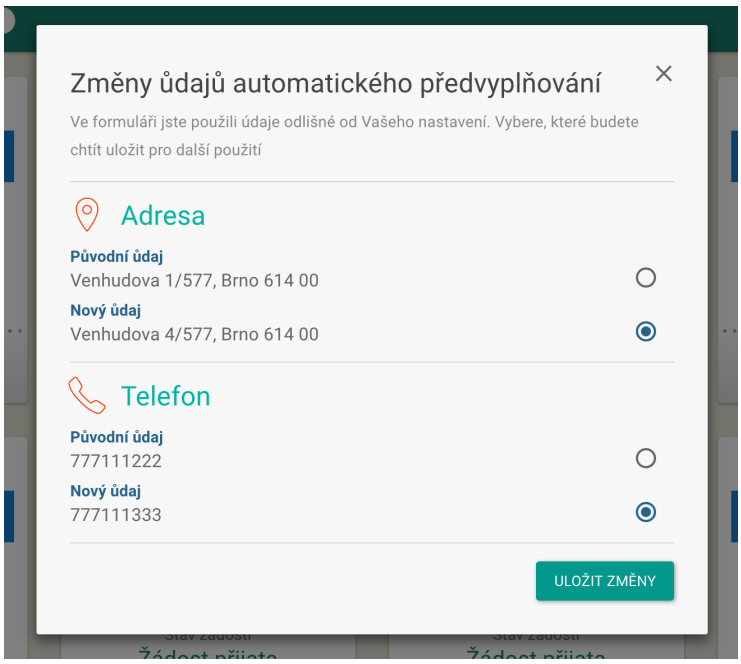
```
{
   "aid": 281474976710690,
   "creationTime": 1479486789411,
   "iid": 281474976710956,
   "msid": 281474976712208,
   "pid": 281474976710673,
   "state": 1,
   "text": {\"type\":\"SET_FORM"\,"label\":\"Dobropis"}"
```

Parametr **aid** určuje, jaký uživatel zprávu vytvořil, **iid** identifikuje reklamaci, **msid** je unikátní ID zprávy, **pid** je identifikátor obchodu, pod který spadá reklamace. Pokud **state** nabývá hodnoty různé od 1, zpráva se nezobrazí. Poslední parametr *text* je obsah zprávy. Zprávy se dělí na systémové a uživatelské. Systémové jsou vytvořeny sami v rámci aplikace a jedná se převážně o notifikace o změnách. Díky těmto zprávám je veškerá historie zaznamenána v chatu. Tato konkrétní zpráva je systémová a jedná se o oznámení o změně typu reklamačního formuláře.

Uživatelské profily

Každý přihlášený uživatel má možnosti si upravit svůj profil. Administrátor může kromě svého profilu upravovat i profily svých pracovníků. Součástí každého profilu je avatar, neboli profilový obrázek uživatele. Pokud uživatel nenahraje fotografii ke svému profilu, je avatar generován automaticky na klientovi. Avatar se generuje na základě aid uživatele. Vygenerovaný avatar není unikátní, více uživatelů může mít stejného avatara. Celkový počet všech takových možných avatarů je 98. V profilu lze navíc změnit heslo, jméno, příjmení, jazyk aplikace a několik dalších položek, které jsou závislé na roli uživatele.

Zákazník má v nastavení profilu sekci **Automatické předvyplňování údajů**. Tyto údaje se předvyplňuji při vytváření nové reklamace. V situaci, kdy se údaje u nově zadané reklamace budou lišit od automatických, aplikace se uživatele dotáže, zda je nechce přepsat a v případě potvrzení je přepíše novými. Vizuální podoba tohoto dotazu je na obrázku 5.8.



Obrázek 5.8: Ukázka aplikace - změna automatických údajů.

Lokalizace

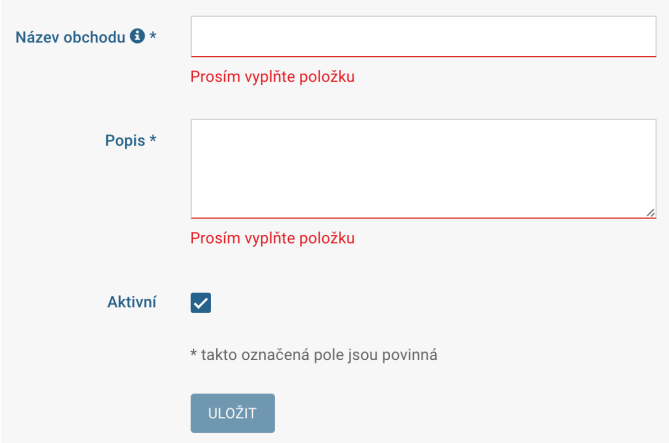
Aplikaci lze přepínat mezi českou a anglickou lokalizací. Lze přidávat i další jazyky. Pro překlad řetězců je vytvořena funkce s názvem skládajícího se ze dvou podtržítek. Tato funkce jako parametr příjme klíč daného řetězce, který se má přeložit. Pokud pro daný klíč a nastavený jazyk najde odpovídající řetězec, tak ho vrátí. V druhém případě vrátí vstupní klíč, což se hodí při vývoji. Přeložené texty se nachází v souboru */src/client/lang.js*. Při překládání textů je možné aplikaci přepnout do speciálního módu, kdy si aktuální překlady stahuje online dle zadaného URL. Přepnutí lze docílit v souboru */src/client/index.js* na řádku 193 přepsáním hodnoty v proměnné `onlineTranslate` z `null` na URL, na kterém se nachází texty. Tento proces je vhodný použít v případě, že texty nevkládá vývojář, ale 3. strana, které nemůže zasahovat do zdrojových kódů.

Formuláře

Jednou z nejpoužívanějších komponent v aplikaci je komponenta formuláře. Tato komponenta se používá téměř na každé stránce aplikace. Její implementace je v adresáři */src/client/components/Form*. Pro deklaraci podoby výsledného formuláře se používá pole objektů, které popisuje strukturu. Použití této komponenty je například na stránce nastavení základních údajů obchodu. Struktura je následující.

```
[
  {id: 'name', type: 'TEXT', label: __('storeSettings.StoreName'),
    required: true, helper: __('storeSettings.StoreNameHelper') },
  {id: 'description', type: 'TEXTAREA', label: __('soreSettings.Desc'),
     required: true},
  {id: 'active', type: 'CHECKBOX', label: __('common.Active')},
]
```

Formulářové prvky se zobrazí ve stejném pořadí, jako je tomu v poli. Každý prvek má povinný atribut **id**. V ukázce lze vidět atribut **label**, který slouží jako popisek prvku. Pro nápovědu je určený atribut **helper**, text se zobrazí až po najetí myši na symbol otazníku. Pokud je povinnost prvek vyplnit, musí se atribut `required` nastavit na hodnotu `true`. Výše je vidět použití funkce pro lokalizaci textů v atributech **label** a **helper**. Formulář podporuje několik typů prvků. To o jaký prvek se konkrétně jedná, je dáno hodnotou atributu **type**. Ostatní parametry jsou závislé na typu prvku. Výsledná podoba formuláře v ukázce je znázorněna na obrázku 5.9. Součástí formuláře je zároveň tlačítko pro uložení, popisek tohoto tlačítka lze zvolit libovolně. Pokud formulář obsahuje chybu, tlačítko je zašedlé.



Obrázek 5.9: Ukázka aplikace - komponenta formuláře.

Kromě pravidla pro povinnost vyplnit položku, existují u některých komponent i automatická validační pravidla. Například u komponenty pro vkládání emailové adresy se automaticky kontroluje formát. Při složitějších validačních pravidlech lze ke komponentě přidat jako parametr vlastní validační pravidla, jenž mohou být libovolná. To se například využívá ve formuláři pro změnu hesla. Formulář obsahuje dva textové vstupy, do kterých se vkládá nové heslo. Druhý vstup je pro kontrolu, zda se uživatel nepřepsal při zadávání nového. V tomto formuláři se kontroluje délka i totožnost hesel. Kód pro validaci vypadá následovně:

```
{
   password: [
      (val) => val && val.length < 6 && __('common.PasswordShort')
   ],
    password_control: [
        (val, values) => val && val !== values.get('password') &&
            __('common.PasswordDifferent')
    ]
}
```

Validační funkce jsou zapsané pomocí tzv. arrow function. Celá struktura je objekt, ve které se jména atributu mapují na identifikátory formulářových prvků. Na každý tento atribut se váže pole, jehož prvky musí být funkce. Formulář při validování tyto funkce volá a na základě jejich návratové hodnoty rozhoduje, zda došlo k chybě nebo ne. Pokud funkce vrátí textový řetězec, znamená to, že validace neprošla a jako chybová hláška se zobrazí vrácený řetězec. Funkce je zavolaná se dvěma parametry a to hodnota aktuálního formulářového prvku a objektu, který obsahuje hodnoty všech formulářových prvků. Důvodem, proč jsou tyto funkce v poli, je to, aby se mohlo přidat pro jeden formulářový prvek více funkcí.

Uživatel typu administrátor má možnost vytvářet vlastní podoby formulářů pro reklamační požadavky. Pro tento účel slouží komponenta */src/client/components/Form/Builder*. Pomocí této komponenty lze sestavit libovolný formulář, až na možnost nastavit libovolná validační pravidla.

Formulářové položky se dělí do dvou skupin. První skupinou jsou vstupní prvky a druhou dekorační. Dekorační prvky slouží pouze jako popisky a nelze do nich vkládat obsah. Formulář má implementované tyto vstupní prvky: bankovní účet, adresa, zaškrtávací box, seznam zaškrtávacích boxů, datum, text, víceřádkový text, jméno, výběr ze seznamu.

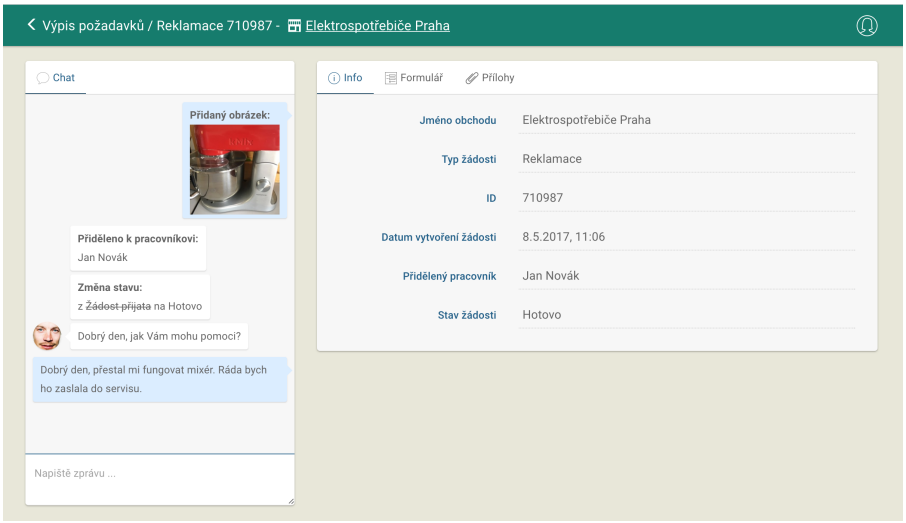
Speciální formulářovou položkou je adresa. Ta využívá API Google Maps pro automatické vyplňování adresy. Uživateli sama našeptává a doplňuje kompletní adresu na základě stručně zapsaných údajů. U adresy je také možné nastavit, které prvky adresy se mají vyplnit a které jsou skryté.

Detail reklamace

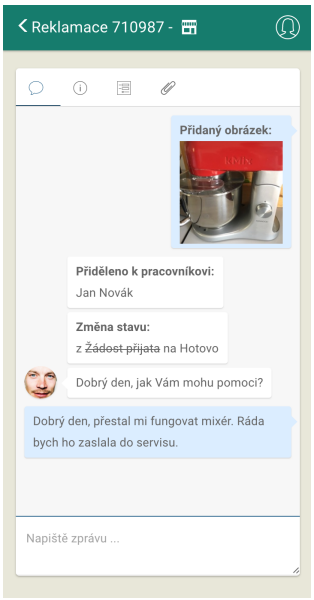
Důležitou části aplikace je samotný detail reklamace. K detailu reklamace se dostane zákazník, který ho vytvořil i zaměstnanci daného obchodu. Pokud má okno větší rozlišení než 991px, stránka má podle návrhu dvě části. Vlevo zobrazuje chat a v pravé obsah v záložkách. Pokud je šířka okna menší, tak se chat přemístí mezi záložky obsahu. Detail má tedy responsivní chování. Je možné ho ovládat i na mobilních telefonech. Na obrázku 5.10 je obrazovka detailu při desktopovém rozlišení. Obrázek 5.11 ukazuje stejnou obrazovku při rozlišení mobilního telefonu.

Na nižším rozlišení navíc zmizí text z menu u záložek a zůstane pouze ikona. Horní lišta se také zestruční - zmizí název obchodu a text u tlačítka, které slouží pro vrácení na všechny požadavky.

Zprávy v chatu jsou zobrazeny v boxech. Barva těchto boxů, zarovnání a zobáček naznačují od koho pochází zpráva. Zprávy přihlášeného uživatele jsou modré zarovnané doprava se zobáčkem směřujícím také doprava. Zároveň u nich není avatar. U zpráv od jiných uživatelů se zobrazuje avatar avšak vždy až u poslední zprávy, kterou uživatel napsal. To lze vidět na obrázku 5.11. Chat kromě klasického zasílání zpráv mezi zákazníkem a zaměstnancem zároveň zobrazuje veškeré změny, ke kterým dochází u dané reklamace. Kliknutím



Obrázek 5.10: Ukázka aplikace - detail reklamace na desktopech.



Obrázek 5.11: Ukázka aplikace - detail reklamace na telefonech.

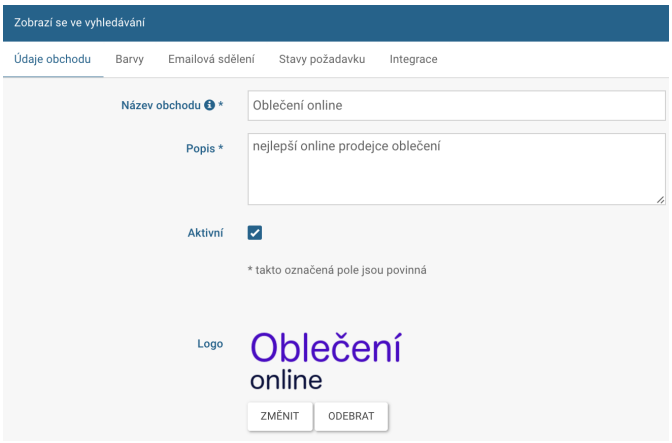
na zprávu se zobrazí datum vytvoření zprávy. Na obrázku 5.11 je ukázka 2 systémových zpráv. První je informace o tom, že uživatel nahrál obrázek a zároveň je zobrazen náhled tohoto obrázku. Pokud by nahraný soubor nebyl obrázek, ale například dokument PDF, jako náhled se zobrazí ikona PDF. Druhou zprávu je informace o změně stavu. Při změnách je zobrazen vždy předchozí a nový stav. Předchozí stav je proškrtnutý, aby znázorňoval, že už neplatí.

5.3 Ukázka použití

V této části je aplikace demonstrovaná na ukázkovém obchodu. Nechť tento obchod prodává oblečení a má 2 zaměstnance. Chce umožnit svým zákazníkům kromě reklamování i výměnu

velikosti zboží. Obchod má pouze 2 zaměstnance. Před zpracováním žádosti je potřeba odsouhlasit smluvní podmínky.

Majitel obchodu se tedy zaregistruje do aplikace a vytvoří si nový obchod. Obchod má i své logo, které si vloží do aplikace v sekci **Základní nastavení**. Na obrázku 5.12 je ukázka podoby takového nastavení.

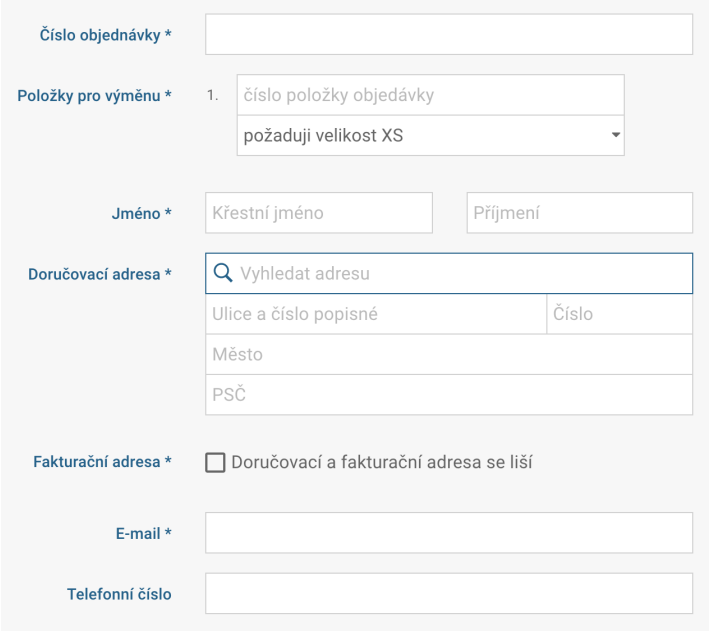


Obrázek 5.12: Ukázkový obchod - základní nastavení.

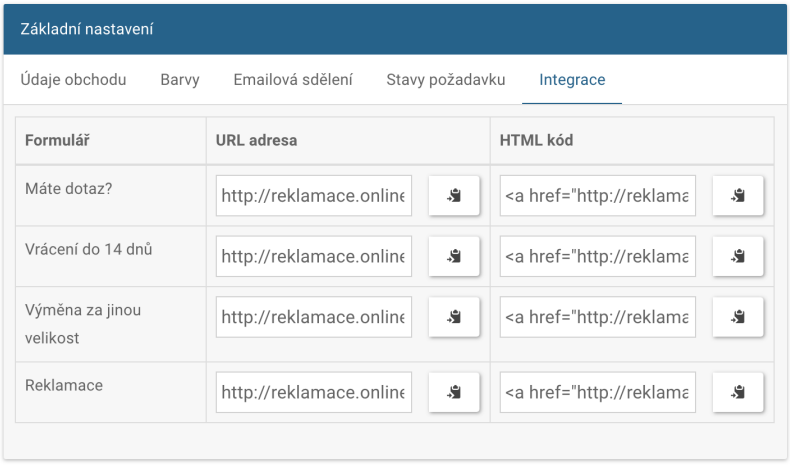
Jak již bylo zmíněno, obchod chce svým zákazníkům umožnit reklamování zboží a jeho výměnu za jinou velikost. Společně s tím chce navíc umožnit vrácení zboží do 14 dnů bez udání důvodu. Z toho plyne, že budou existovat 3 typy formulářů. Pro každou žádost se totiž formulář bude mírně lišit. Obchod bude po zákazníkovi vždy požadovat číslo objednávky, číslo položky objednávky, jméno, příjmení, email, telefon a adresu. Toto budou mít všechny formuláře společné. Formulář reklamace bude navíc požadovat, aby zákazník popsal závadu. U formuláře pro výměnu za jinou velikost bude potřeba zadat novou velikost. Pro možnost vrátit do 14 dnů není požadovaný žádný důvod, a proto tento formulář bude obsahovat pouze předchozí základní prvky.

Všechny formuláře jsou si podobné. Administrátor tedy vytvoří v sekci **Formuláře zákazníka** nejprve formulář pro vrácení zboží do 14 dnů. Potom vytvoří zbývající tak, že 1. formulář bude sloužit jako šablona pro ostatní, kde pouze doplní zbývající položky. Vzhled nastaveného formuláře pro výměnu zboží je na obrázku 5.13, na kterém je jako druhý vstupní parametr seznam položek objednávky. Zákazník může pomocí tohoto vložit libovolný počet položek objednávky a ke každé z ní přiřadit za jakou velikost chce zboží vyměnit. V této sekci administrátor zároveň přiřadí smluvní podmínky, které nahraje jako dokument PDF.

Následně administrátor musí nastavit pracovníky. V sekci **Pracovníci zákaznické podpory** postupně vytvoří nové profily zaměstnanců. Pro každého z nich musí zadat unikátní emailovou adresu a vymyslet přihlašovací heslo. Po vytvoření tyto údaje sdělí svým zaměstnancům. Poté musí v nastavení obchodu v sekci **Zákaznická podpora** přiřadit zaměstnance k obchodu. V posledním kroku je potřeba u obchodu v části **Integrace** zkopírovat URL adresy formulářů, které přidá na svoji stránku obchodu. Tuto stránku lze vidět na obrázku 5.14. Kromě zmíněných 3 formulářů je v tabulce vidět i další formulář, který je určený k dotazu zákazníka.

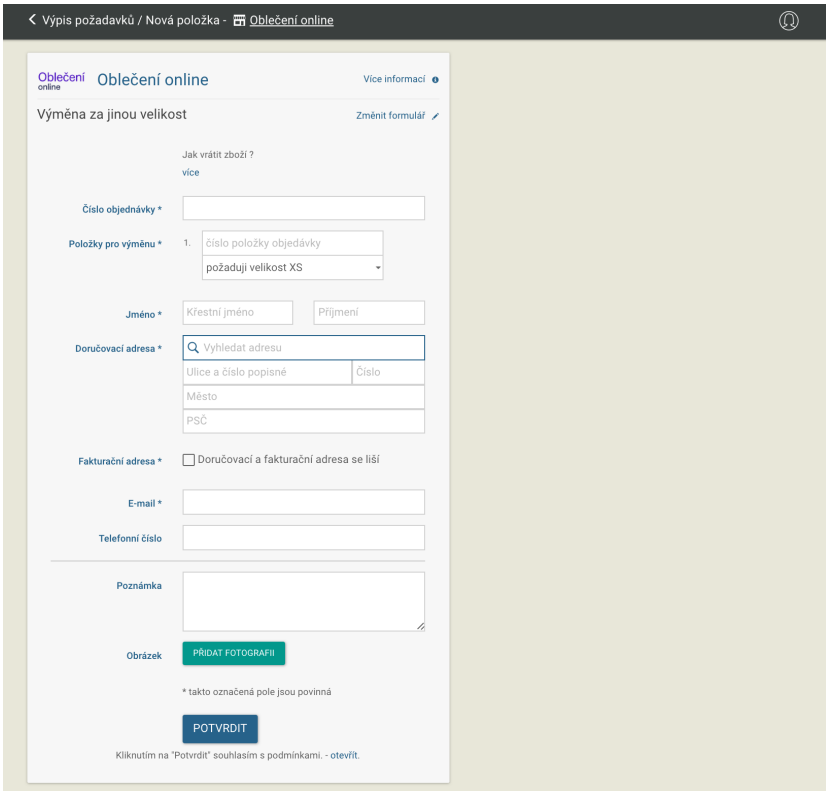


Obrázek 5.13: Ukázkový obchod - formulář.



Obrázek 5.14: Ukázkový obchod - integrace.

Zákazník, který by chtěl vyměnit zboží, může do systému přistoupit přes odkaz na stránkách obchodu. Po načtení aplikace se mu na stránce zobrazí daný formulář společně s názvem a logem obchodu. Tato stránka je na obrázku 5.15.



Obrázek 5.15: Ukázkový obchod - nová žádost.

5.4 Testy

Pro ověření správné funkčnosti aplikace před jejím produkčním nasazením je vhodné aplikaci otestovat. Některé testy je třeba dělat opakovaně při každé změně aplikace. Zde je soupis několika typů testů.

Kompatibilita zařízení a prohlížečů

Na různých zařízeních a prohlížečích může aplikace fungovat rozdílně. Některé typy prohlížečů nemusí podporovat veškeré prvky, které aplikace využívá. Ovšem ne všechny věci jsou pro fungování aplikace nutné. Některé jsou pouze vzhledového charakteru, a proto nemusí být vždy problém, když nefungují správně. Je náročné určit přesnou hranici toho, která věc je již zásahem do fungování, když to, že nefunguje, neomezuje možnosti aplikace. Další hranicí, kterou je potřeba si určit je to, pro jak velké množství prohlížečů a zařízení chceme aplikaci testovat. Poslední hranicí je rozsah toho, jak moc podrobně se aplikace testuje. Vyzkoušet všechny možnosti v aplikaci může být časově velice obtížné. Problémem také může být to, že pro testování uživatel nemá dostatek zařízení a přístup k různým operačním systémům a verzím prohlížečů. Tento problém se dá ovšem řešit pomocí virtualizace, ať už vytvořením své vlastní nebo využitím online nástrojů. Po určení hranic se aplikace může začít postupně testovat tak, že se zkouší její funkčnost. Aplikace se dá testovat manuálně nebo lze vytvořit automatické testy. Vytvoření automatických testů může být náročné, neboť může být potřeba, aby test sám rozpoznával, kde se například zobrazil prvek, jak moc je viditelný a tak dále. U automatických testů se někdy používá i porovnávání obrazu.

Pro aplikaci, která je řešená v rámci této diplomové práce, by bylo vhodné provést testy i na starších prohlížečích, neboť aplikace může přebírat určitou zodpovědnost. Pokud by u ní docházelo k závažné chybě, mohla by například zdržet zaměstnance nebo znemožnit reklamování některému zákazníkovi. To by firmu mohlo stát peníze. Otestovat aplikaci by tedy bylo nad rámec této práce z důvodů časové náročnosti. Pokud by se aplikace prezentovala tím, že je plně funkční na mobilních telefonech, bylo by třeba testovat i velké množství mobilních zařízeních s různými verzemi softwaru.

Návrhem testu pro desktopové systémy je testování aplikace na operačních systémech Windows XP, Windows 7, Windows 8.1, Windows 10, macOS Sierra a macOS El Capitan. Z toho důvodu, že je v České Republice nejpoužívanější právě Windows, měl by na něj být kladen největší důraz. Na těchto operačních systémech je potom navrhováno vyzkoušet tyto prohlížeče (pokud je podporují) Chrome, Internet Explorer 10, Internet Expoler 11, Edge, Safari, Mozilla Firefox, Opera a to v nejaktuálnějších verzích.

V rámci diplomové práce by takto otestovat aplikaci bylo časově velice náročné. Aplikace byla testovaná v prohlížečích Chrome, Mozilla Firefox a Safari na operačních systémech MacOS Sierra a Windows 10. Testováním se ověřilo, že aplikace v těchto prohlížečích fungovala.

Uživatelské testy

Důležitým testem je i to, jak se s aplikací dokáží seznámit uživatelé a jak rychle dokáží plnit typické úkoly. Díky tomu lze odchytit chyby v návrhu uživatelského prostředí a chování aplikace. Z důvodu existence 3 rolí, je třeba 3 rozdílných testů, specifických pro každou roli.

Test lze provést vytvořením testovacího protokolu, který obsahuje popis typických úkolů v aplikaci. Skupině uživatelů se poté tyto úkoly zadají a sleduje se jejich chování a rychlost plnění. Poznatky se zaznamenají. Lze si například všimnout, že uživatel něco hledá v jiné části aplikace, než by měl. Skupinu uživatelů je vhodné vybrat co největší a nejpestřejší, tím je myšleno různé pohlaví, věk, vzdělání a podobně.

Pro lepší pochopení toho, kterých prvků si uživatelé všímají a kterých ne, je vhodné do aplikace integrovat tzv. mouse tracking. Ten slouží k pozorování pohybu cursoru po obrazovce. Výsledkem testu je teplotní mapa, která pomocí barev znázorňuje, jak moc se na konkrétních místech uživatel pohyboval myší. Lepší variantou je použití eye trackingu neboli sledování pohybu očí. Ne vždy uživatel najede kursorem na místo, na které se dívá očima. Eye tracking je oproti mouse tracking finančně a časově náročnější.

Pro aplikaci byly navrženy 3 testy, každý pro jednu roli. Tyto testy byly navrženy tak, aby byly otestovány typické úkony s aplikací. Dílčí části testů na sebe navazují. Jedná se tedy o 3 testy, přičemž každý z nich je rozdělen na několik částí.

Výsledky testů pro zákaznickou roli jsou v tabulce 5.1. Během testování bylo vypozorováno, že uživatelé byli zaskočeni výpisem zákaznických požadavků. Naměřené časy přehození typu reklamace byly vysoké převážně z toho důvodu, že uživatelé četli dlouhý text, který je upozorňoval na to, že jejich předchozí formulář bude přepsán.

Tabulka 5.2 znázorňuje výsledky testů na zaměstnancích. U části, ve které měli uživatelé nastavit sebe jako zaměstnance, nevyužívali tlačítko k tomu určené. Bylo by třeba na to uživatele lépe upozornit.

Poslední testovanou rolí byl administrátor. Tato role je oproti ostatním nejméně používaná, zároveň je však nejsložitější. Výsledky jsou v tabulce 5.3. Při testování bylo zaznamenáno, že uživatelé mají problém najít URL k formulářům. Nejspíše je to dané špatným

Popis testu	Aritmetický průměr
Vytvoření nové žádosti typu Dotaz	11,63
Přidání k žádosti fotografii	12,32
Napsat zprávu s textem "Dobrý den"	5,82
Přehodit žádost na typ Reklamace	24,91
Zjistit číslo reklamace	3,34

Tabulka 5.1: Testy na zákaznicích.

Popis testu	Aritmetický průměr
Vyfiltrovat si žádosti o které se nikdo nestará	14,19
U té nejstarší nastavit sebe jako zaměstnance	30,74
Do interní poznámky napsat "Poznámka"	9,27
Změnit žádost na stav hotovo	12,93
Přidat si obrázek ke svému profilu	11,71
K předchozí žádosti přidat zprávu "Zpráva"	13,54

Tabulka 5.2: Testy na zaměstnancích.

označením záložky. Většina testovaných uživatelů se k tomuto URL náhodou proklikala, když zkoušela všechny možné záložky aplikace.

Testy byly provedeny na 10 uživatelích. Pro reálné nasazení je to ovšem málo a bylo by potřeba každou roli otestovat alespoň na 100 uživatelích. To je ovšem z časových důvodů nad rámec této práce.

Konektivita

Mezi požadavky na výslednou aplikaci byla podpora na mobilních zařízeních. Důvodem bylo, aby zákazník či zaměstnanec mohli aplikaci efektivně používat i mimo svůj domov. Uživatel může být odkázán na mobilní internet. Je tedy nutné počítat i s tím, že v takovém případě může být rychlost znatelně menší. Tento test se zaměřuje na to, jak dlouho trvá načtení aplikace na různých rychlostech připojení. Aplikace si při prvotním načtení stáhne veškerou aktuální databázi dat a všechny potřebné soubory pro fungování aplikace. Při dalších načteních již stahuje pouze databázi. Při práci v aplikaci se pak již stahují pouze změny v databázi a obrázky.

Pro testování byl použit simulátor internetové konektivity, který je součástí prohlížeče Google Chrome. Pro testování zákazníka byly vytvořeny testovací data, která zahrnovala

Popis testu	Aritmetický průměr
Vytvoření nového obchodu	17,49
Přidat do obchodu nový formulář	18,17
Obchodu i formulář aktivovat	35,52
Zjistit URL k formuláři	38,20
Vytvořit nového zaměstnance	28,98
Přiřadit zaměstnance k obchodu	15,59
Vstoupit od obchodu jako zaměstnanec	17,21

Tabulka 5.3: Testy na administrátorech.

Typ	Rychlost	Latence	Zákazník	Zaměstnanec
GPRS	50 kb/s	500 ms	5,3 min / 2.14 s	5,3 min / 2,3 s
Obvyklé G2	250 kb/s	300 ms	1 min / 1,92 s	1 min / 2,10 s
Dobré G2	450 kb/s	150 ms	35,94 s / 1,21 s	36,22 s /1,44 s
Obvyklé G3	750 kb/s	100 ms	22,33 / 1,18 s	21,29 /1,43 s
Dobré G3	1,5 Mb/s	40 ms	11,38 / 1,12	11,42 s / 1,39 s
G4	4,0 Mb/s	20 ms	8,2 / 1,02	8,09 s / 1,08 s

Tabulka 5.4: Výsledky testů konektivity.

4 žádosti. U testování zaměstnance bylo ukázkových dat více. Nejprve byla otestovaná rychlost prvního spojení, kdy uživatel neměl žádná data v paměti prohlížeče. Poté bylo měřeno, jak dlouho potrvá druhé načtení, kdy už se stahuje pouze databáze. Zákazník stahoval při prvním načtení 1,9 MB. U druhého načtení se stahovalo 680B. U zaměstnance je velikost dat 1,9 MB a u druhého načtení 24,5 KB. Rozdíl je pouze ve velikosti databáze.

V tabulce 5.4 jsou naměřené hodnoty. Pro každý typ proběhly 3 měření, které byly aritmeticky zprůměrovány. Z časů lze vyčíst, že mezi prvním a druhým načtením je obrovský rozdíl. Naproti tomu rozdíl mezi zákazníkem a zaměstnancem není téměř žádný. Důvodem je, že se stahuje téměř stejné množství dat. Použití při GPRS nebo G2 síti se aplikace poprvé načítá velice dlouho, avšak v dnešní době se tyto sítě moc nepoužívají. Rychlost není závislá pouze na připojení klientského zařízení. Časy ovlivňuje i připojení a celková rychlost serveru a sítě mezi klientem a serverem.

Rychlost je převážně závislá na velikosti dat, které se přenáší. Tyto data by se daly zmenšit tím, že by se rozdělil výsledný kód aplikace na více části. Ty by se stahovaly průběžně dle potřeby.

Kapitola 6

Závěr

Cílem práce bylo vytvořit aplikaci pro snadné vyřizování reklamací za použití moderních technologií. Výsledná aplikace měla sloužit obchodům jako systém, přes který by umožnily svým zákazníkům vyřizovat reklamace. Tyto požadavky by pak následně zaměstnanci obchodu spravovali.

Ze začátku práce jsem prostudoval přístup tvorby webových aplikací a technologie, které jsou pro vytváření využívány. Byly popsány techniky a nástroje, jenž vývojový proces i produkt zlepšují. Zaměřil jsem se především na klientskou část, konkrétně na jazyk JavaScript a nástroje Webpack a React. Důraz byl kladen i na tvorbu uživatelského prostředí pomocí frameworků k tomu určených. Vyzkoušel jsem a popsal, jakým způsobem reklamace řeší některé velké české e-shopy. Poté byly shrnuty současné technologie a bylo vyvozeno, že se některé z daných technologií hodí pro rychlou dynamickou aplikaci, která by měla obsluhovat reklamační požadavky.

Na základě nastudovaných postupů jsem navrhl a implementoval klientskou část a komunikaci se serverem. Jako koncept jsem zvolil Single Page Application. Pro implementaci jsem využil technologii React společně se stavovým kontejnerem Redux. Do spolupráce se zapojila firma Awexa s.r.o, která navrhla a implementovala serverovou část aplikace na základě mých požadavků na služby.

Použití vytvořené aplikace je demonstrováno na ukázkovém obchodu. Poté jsem pro aplikaci navrhl řadů testů a ty, které byly v mých možnostech, jsem i provedl. Testy byly zaměřené na práci s uživatelským prostředím, podporu různých zařízení a závislosti na rychlosti připojení. Bylo ověřeno, že na klasických moderních prohlížečích lze aplikace používat. Stejně tak na mobilních telefonech, jenž využívají pro mobilní data sítě typu G4 a vyšší nebo jsou připojeny přes WiFi. Před produkčním nasazením aplikace by bylo potřeba testy udělat velice důkladně na co nejvíce zařízeních, toto je ovšem nad časový rámec této diplomové práce.

Mnou vytvořená aplikace dává možnost obchodům zaregistrovat se do systému a umožnit jejich zákazníkům snadné vyřizování reklamace. Obchody mají možnost si aplikaci v mnoha ohledech nakonfigurovat dle svých požadavků, a to od grafických úprav až po funkční prvky. Zákazníci mají možnost se zaměstnanci chatovat a zasílat si přílohy. Zákazníci i zaměstnanci mohou aplikaci pohodlně používat i na mobilních telefonech a tabletech bez ztráty na funkčnosti.

Aplikace byla vytvářena tak, aby šla lehce rozšířit s využitím již hotových komponent. Vytvořit novou stránku by pravděpodobně znamenalo jen pospojovat již hotové komponenty, podobně jako to dělají některé frameworky. Rozšířit aplikaci by šlo například o přidání náhledu k více typům multimediálních souborů. Dalším přínosem by byla možnost se

přihlásit pomocí některé z běžných sociálních sítí a ulehčit tak uživateli registraci. V neposlední řadě by bylo přínosné do chatu doplnit status o tom, zda je uživatel online a zda zrovna píše zprávu. Zaměstnanci by se dokonce mohla rozepsaná zpráva od zákazníka rovnou zobrazovat, aby lépe pochopil její obsah.

Literatura

[1] Abramov, D.; Clark, A.: *Redux - Dokumentace*. GitHub Inc., [Online; navštíveno 25.11.2016].
URL https://github.com/reactjs/redux/tree/master/docs

[2] Banks, A.; Porcello, E.: *earning React: Functional Web Development with React and Flux*. O'Reilly Media, 2014, ISBN 978-1491954621.

[3] Duckett, J.: *HTML and CSS: Design and Build Websites*. John Wiley & Sons, Inc., 2011, ISBN 978-1118008188.

[4] Duckett, J.: *JavaScript and JQuery: Interactive Front-End Web Development*. John Wiley & Sons, Inc., 2014, ISBN 978-1118531648.

[5] Florence, R.; Jackson, M.: *React router - Dokumentace*. GitHub Inc., [Online; navštíveno 10.11.2016].
URL https://github.com/ReactTraining/react-router/tree/master/docs

[6] Google: *Material Design*. GitHub Inc., [Online; navštíveno 10.11.2016].
URL https://material.io/

[7] Hampton Catlin, N. W.; Eppstein, C.: *SASS - Dokumentace*. Sass, [Online; navštíveno 2.11.2016].
URL http://sass-lang.com/documentation/

[8] Inc, F.: *Immutable.js - Dokumentace*. GitHub Inc., [Online; navštíveno 25.11.2016].
URL https://facebook.github.io/immutable-js/docs/

[9] Inc, F.: *React - Dokumentace*. GitHub Inc, [Online; navštíveno 24.10.2016].
URL https://facebook.github.io/react/docs/

[10] Mew, K.: *Learning Material Design*. Packt Publishing - ebooks Account, 2015, ISBN 978-1785289811.

[11] Michálek, M.: *Atomický design a Pattern Lab: návrat do budoucnosti návrhu uživatelských rozhraní*. Devel.cz Lab s.r.o., [Online; navštíveno 28.11.2016].
URL https://www.zdrojak.cz/clanky/atomicky-design-pattern-lab-navrat-budoucnosti-navrhu-uzivatelskych-rozhrani/

[12] Michálek, M.: *Vzhůru do CSS*. V Zeleném údolí 1316/12 148 00, Praha, 2016.

[13] Mike Cantelon, N. R., Marc Harter: *Node.js in Action*. Manning Publications, 2014, ISBN 978-1617290572.

[14] Mikowski, M. S.; Powel, J. C.: *Single Page Web Applications*. Manning Publications Co., 2014, ISBN 978-1617290756.

[15] Saternos, C.: *Client-Server Web Apps with JavaScript and Java: Rich, Scalable, and RESTful*. O'Reilly Media, 2014, ISBN 978-1449369330.

[16] Spurlock, J.: *Bootstrap: Responsive Web Development*. O'Reilly Media, 2013, ISBN 978-1449343910.

[17] Stefanov, S.: *React: Up & Running: Building Web Applications*. O'Reilly Media, 2015, ISBN 978-1491931820.

Přílohy

Příloha A

Obsah CD

- /text/ - technická zpráva
- /client/ - zdrojový kód aplikace
- /poster.png - plakát
- /video.mp4 - video
- /tests/ - testovací protokoly

Příloha B

Plakát