VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PORTÁL PRO ADMINISTRACI ČASOSBĚRNÝCH SYSTÉMŮ

PORTAL FOR THE ADMINISTRATION OF TIME-LAPSE SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ DACER**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. PAVOL KORČEK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Dacer Ondřej**
Program: Informační technologie
Název: **Portál pro administraci časosběrných systémů**
Portal for the Administration of Time-Lapse Systems
Kategorie: Web

Zadání:

1. Nastudujte vybraný framework pro vývoj responzivních webových aplikací.
2. Seznamte se se systémem pro ovládání časosběrných systémů a jeho aplikačním rozhraním.
3. Navrhněte webové responzivní uživatelské rozhraní pro tento systém. Při návrhu uvažujte možnost konfigurovatelných pohledů na aplikaci.
4. Ve zvolené technologii implementujte takto navržený systém.
5. Implementovaný systém otestujte.
6. Zhodnoťte dosažené výsledky práce a diskutujte další možná rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Korček Pavol, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem této práce je vytvoření administračního portálu pro časosběrný systém. Při řešení práce bylo analyzováno několik frameworků a knihoven vhodných pro tvorbu responzivních webových aplikací, testovací framework Jest a komunikační protokol WebSocket. Dále bylo nezbytné se seznámit s existujícím časosberným systémem a s jeho požadavky na administrační portál. Výsledná webová aplikace byla implementována pomocí javascriptové knihovny React. Využitím protokolu WebSocket bylo umožněno aktualizovat data a zobrazovat serverové notifikace v reálném čase. Na závěr byla celá aplikace řádně otestována a nasazena.

Abstract

The goal of this thesis is a creation of administration portal for time-lapse system. In this thesis were analysed several frameworks and libraries suitable for responsive web applications development, testing framework Jest and communication protocol WebSocket. It was also necessary to get familiarized with existing time-lapse system and with its requirements for administration portal. Final web application was implemented with Javascript library React. With the use of WebSocket protocol was made real-time data update and server notifications display possible. Finally, the entire application was properly tested and deployed.

Klíčová slova

Responzivní webová aplikace, JavaScript, React, Jest, WebSocket, BIXION

Keywords

Responsive web application, JavaScript, React, Jest, WebSocket, BIXION

Citace

DACER, Ondřej. *Portál pro administraci časosběrných systémů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Pavol Korček, Ph.D.

Portál pro administraci časosběrných systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavola Korčeka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ondřej Dacer
11. května 2021

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Pavolu Korčekovi Ph.D. za vstřícné a odborné vedení. Dále bych chtěl poděkovat také Ing. Josefu Hájkovi a Ing. Davidu Gargulákovi za technické konzultace, ochotu a trpělivost při řešení praktické části této bakalářské práce.

Obsah

Kapitola 1

Úvod

Sledování vývoje prací na stavbě či monitorování prostředí patří k základním využitím časosběrných systémů. Rozvoj lidské společnosti přispěl k tomu, že v některých případech může být fyzická přítomnost člověka nahrazena technickým řešením. Portál pro administraci časosběrného systému totiž svým uživatelům umožňuje sledovat dění ve vzdálených destinacích pohodlně a prakticky odkudkoliv. Informace o dané lokalitě může mít uživatel k dispozici rychle a přehledně na jednom místě, konkrétně na displeji zařízení s připojením k Internetu.

Při návrhu uživatelského rozhraní nejenom administračních portálů je nutné dbát několika zásadních pravidel, které uživateli zajistí, že bude moci pracovat intuitivně, jednoduše, rychle a efektivně. S ohledem na skutečnost, že se v posledních letech zvyšuje trend návštěvnosti webových aplikací z mobilních zařízení, je nutné myslet i na tato malá zařízení. Tedy na to, aby byla webová aplikace responzivní.

Hlavním cílem této práce je navrhnout a vytvořit funkční, responzivní a uživatelsky přívětivý portál pro administraci časosběrného systému společnosti BIXION s. r. o. Výběr technologií pro implementaci portálu bude založen na podrobné analýze, stejně jako jeho následné testování. Výsledný portál by měl do budoucna umožňovat integraci dalších rozšíření. Dále by měl být také natolik modulární, aby se daly některé jeho části využít pro práci na projektech podobného typu.

Následující kapitola 2 představuje pojem responzivní design a zařazuje jej do historického kontextu. Stejná kapitola dále vysvětluje rozdíl mezi pojmy framework a knihovna. Obsahuje také analýzu vybraných frameworků a knihoven, které jsou porovnávány na základě popsaných kritérií. Technologiím, které byly využity k implementaci a testování výsledné webové aplikace, jsou věnovány kapitoly 3 a 4. Kapitola 5 se věnuje popisu využívaného časosběrného systému a požadavkům na administrační portál, jehož řešení je navrženo v téže kapitole. Implementační část bakalářské práce a její testování se nacházejí v kapitolách 6 a 7. V závěrečné kapitole 8 jsou poté shrnuty dosažené výsledky a výstupy této práce. Také jsou zde uvedena rozšíření, která byla implementována nad rámec zadání.

Kapitola 2

Analýza frameworků a knihoven

Na poli vývoje webových aplikací se v dnešní době nachází spousta frameworků a knihoven pro vývoj responzivních webových aplikací, minimálně v řádu několika desítek. Každý z nich má své výhody i nevýhody a hodí se pro řešení rozdílných problémů. Nelze tedy jednoznačně říci, který z nich je obecně nejvhodnější k používání. V této kapitole je provedena analýza vybraných frameworků a knihoven, na jejímž základě byla vybrána technologie, která je detailně popsána v kapitole 3 a která byla použita pro implementaci portálu pro administraci časosběrných systémů.

2.1 Responzivní design

Pojem responzivní design je znám teprve od roku 2010. Vymyslel jej americký web designer Ethan Marcotte a poprvé se objevil jako titulek jeho článku na blogu zaměřeném na tvorbu webových stránek *A List Apart*. Svou tehdejší jednoduchou myšlenku o responzivním designu rozdělil do tří částí: vytvoření pružného rozložení stránky, jeho doplnění pružným obsahem, především obrázky, a přidání podmínek pro změnu rozložení [15].

Hlavní vlna trendu použití responzivního designu pro webové stránky přišla především s rozmachem používání Internetu na mobilních zařízeních. Dnes je již téměř nemyslitelné použití zastaralého statického webu. Z měření [19] například v České republice vyplývá, že byl na podzim roku 2019 počet reálných uživatelů webových stránek na mobilních zařízeních téměř stejný, jako na počítačích. Zároveň časem přibývá mnoho nových druhů zařízení a s nimi i rozlišení, velikost a poměr stran jejich obrazovek od chytrých hodinek, přes displej chytrých domácích spotřebičů, až po velkoformátový panel televize s připojením k Internetu. Nepřizpůsobení webových stránek pro tato zařízení může vést v lepším případě k odlivu některých návštěvníků v důsledku špatné uživatelské zkušenosti. V horším případě k naprosté nepoužitelnosti webových stránek na jakémkoliv jiném zařízení, než je osobní počítač. Dnešní moderní vyhledávače, jako například Google, jsou navíc schopny rozpoznat, zda je stránka vhodná pro mobilní zařízení a v návaznosti na toto kritérium ji umístit mezi přední výsledky, nebo naopak na některé z posledních míst [4]. To vážně ovlivňuje úspěšnost webových stránek z hlediska počtu návštěv, což může vést například k velmi zásadní ztrátě na zisku.

V responzivním designu tedy jde o správné zobrazení webových stránek na každém zařízení. Toho lze v zásadě docílit dvěma způsoby. První možností je použití flexibilní mřížky ve stylech stránky, kde je velikost všech prvků dána pomocí relativních jednotek vzhledem k rozlišení celého okna prohlížeče. Obsah stránky je rozdělen na pomyslnou tabulku

o několika řádcích a sloupcích, k čemuž se využívá vlastnosti kaskádových stylů `display` s hodnotou `grid`, nebo případně `inline-grid`. Poté se definuje kolik řádků a sloupců bude daný prvek v daném rozlišení zabírat. Výhodou je, že lze stejné prvky zobrazovat v rozdílném rozložení na různých zařízeních. S tímto rozložením zároveň souvisí přizpůsobení webových stránek orientaci zařízení. Rozlišujeme orientace *landscape*, kdy je zařízení orientováno na šířku, a *portrait*, kdy je zařízení orientováno na výšku. Webový prohlížeč by měl rozpoznat, ve které orientaci se zařízení nachází a zobrazovaná webová stránka by tak měla přizpůsobit rozložení svých prvků.

Jako druhou možnost lze využít tzv. *media queries*, viz zdrojový kód 2.1. Na základě šířky okna zařízení se celé spektrum rozlišení rozdělí do několika částí, které se oddělí tzv. *breakpointy*. Každá z těchto částí má speciálně přizpůsobené styly tak, aby se webová aplikace na každém zařízení zobrazila co nejpříznivěji. Je samozřejmě možné a mnohdy i výhodné použít oba přístupy zároveň. Vhodným přístupem je také používání relativních jednotek, například pro písmo, které může měnit svou velikost na základě velikosti okna prohlížeče nebo na základě rodičovského elementu.

```
// CSS pravidlo pro media typu obrazovky, jez je sirsi nez 768 pixelu
@media only screen and (min-width: 768px) {
    p { color: black }
}
// CSS pravidlo pro media typu obrazovky, ktera jsou orientovana na vysku
@media only screen and (orientation: portrait) {
    p { font-size: 1em }
}
```

Zdrojový kód 2.1: Ukázka použití *media queries*.

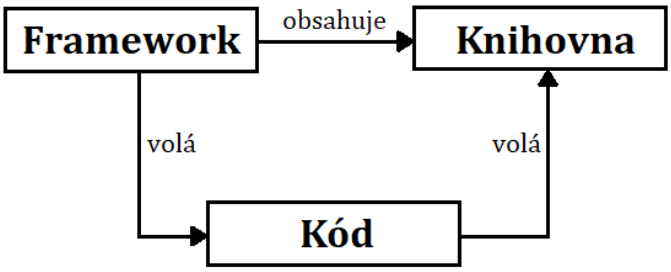
Další důležitou součástí optimalizace stránek pro mobilní zařízení je rychlost načítání a datová náročnost. Zatímco na osobním počítači se stabilním a neomezeným připojením k Internetu načítání obrázků ve vysoké kvalitě ve většině případů nezpůsobí vážné problémy, uživatel mobilního zařízení s omezenými mobilními daty by za vyčerpání jeho limitu příliš vděčný nebyl. Tím spíš, pokud se v dnešní rychlé době stránka načítá příliš dlouho. Těmto problémům se dá čelit například přízpůsobením kvality obrázku podle typu zařízení nebo tzv. *lazy loadingem*. To je technika, jež zařídí načtení části obsahu stránky až v momentě, kdy ji uživatel potřebuje [16].

2.2 Framework versus knihovna

Ještě před samotným porovnáním je nutné definovat, co pojem framework znamená, neboť je často významově zaměňován s pojmem knihovna. Laicky lze říci, že knihovna i framework jsou znovupoužitelné kusy kódu, které programátorům usnadňují práci a umožňují jim tak soustředit se na řešený problém.

Dle [14] lze knihovnu definovat jako soubor procedur a funkcí, v objektovém programování také tříd, který poskytuje jistý okruh služeb. Knihovna musí splňovat několik základních požadavků. „Knihovna si musí rozumět s prakticky libovolnou aplikací, musí být schopna koexistence s dalšími knihovnami a nesmí především ovlivňovat globální stav programu – její vliv musí být lokalizovaný. Součástí toho je, že nesmí provádět tzv. *policy decisions* — říkat aplikaci, co se jak bude dělat.“

Právě *policy decisions* jsou zásadním rozdílem mezi frameworkem a knihovnou. Při použití knihovny má chod aplikace na starost programátor, který určuje kde a kdy knihovnu volat. Zatímco při použití frameworku má na starost chod aplikace právě samotný framework, viz. obrázek 2.1. Framework stanovuje architekturu aplikace a způsob, jakým je aplikace vyvíjena.



Obrázek 2.1: Vztah frameworku a knihovny ke kódu programátora.

2.3 Porovnávaná kritéria

Zvolené frameworky a knihovny byly objektivně zhodnoceny dle několika důležitých kritérií:

- **Dokumentace** – kvalitně sepsaná, přehledná a aktualizovaná dokumentace je nezbytným pomocníkem každého vývojáře, ať už je v dané problematice zběhlý nebo ne. Dokumentace včetně doplňující literatury usnadňuje práci a umožňuje udržovat kód webové aplikace aktuální (anglicky *up-to-date*).
- **Podpora a velikost komunity** – při vývoji nejenom webových aplikací není neobvyklé, že se vyskytne nějaký zdánlivě nevyřešitelný problém, který není v dokumentaci zmíněn. V takovém případě přijde vhod komunita vývojařů či přímo přispěvatelů do zdrojového kódu příslušného frameworku, resp. knihovny. Tato komunita je schopna rychle a efektivně poradit, například na některých za tímto účelem vytvořených webech, mezi které patří Stack Overflow[1].
- **Udržitelnost** – s předešlým kritériem souvisí i udržitelnost. Je nutné zvolit takový framework či knihovnu, jenž se neustále vyvíjí, je aktualizován s ohledem na zabezpečení a jenž má, a s největší pravděpodobností také bude mít, podporu ve všech prohlížečích i v budoucnu.
- **Licence** – zvolená technologie by měla být *open-source*, což znamená, že jej lze zdarma používat, upravovat a také šířit výsledné produkty.
- **Udržitelnost kódu a testování** – je žádoucí mít možnost dekomponovat kód na znovupoužitelné a snadněji udržovatelné komponenty, které lze testovat.
- **Rychlost a výkon** – jedním z hlavních požadavků na nově vzniklý administrační portál je rychlost načítání, proto bude vytvořen jako *single-page* webová aplikace. Taková aplikace totiž nevyžaduje znovu načítání jednotlivých stránek při jejím používání. Většina zdrojů je načtena pouze jednou při vstupu do aplikace. Výhodou

[1] https://stackoverflow.com/

single-page aplikací také je, že se díky vývojařským nástrojům v prohlížečích snadno odlaďují a také že mohou efektivně využívat lokální úložiště prohlížečů. Aplikace může na server zaslat například jeden dotaz a všechna data si uložit přímo v prohlížeči, což urychluje a usnadňuje práci se serverovými daty.

2.4 Srovnání frameworků

V této podkapitole jsou popsány a porovnány celkem 3 populární open-source frameworky, resp. knihovny, všechny vydávané pod licencí MIT[2], nejenom na základě výše zmíněných kritérií, ale i specialit a zvláštností každé technologie. Analýze byly podrobeny dva javascriptové frameworky Angular a Vue a javascriptová knihovna React.

Angular[3]

Současná druhá verze javascriptového frameworku Angular, založeném na TypeScriptu, byla vydána v září roku 2016 společností Google. Vznikla kompletním přepsáním původního webového frameworku AngularJS z roku 2010 od stejné společnosti, ovšem bez možnosti zpětné kompatibility, což vyvolalo vlnu nevole, protože stávající uživatelé původního frameworku tehdy nemohli upgradovat své webové aplikace, a museli tak celou aplikaci od základu přepsat.

V porovnání s dalšími dvěma subjekty má Angular nejstrmější učící křivku. Angular, založený na softwarové architektuře MVC (*Model-View-Controller*), je totiž velmi komplexní a robustní. To se týká i složité, pro začínající vývojaře v Angularu také mnohdy matoucí, dokumentace. Komplexnost a složitost celého frameworku se projevuje také svou datovou velikostí. S 566 KB (v minifikované podobě) je Angular téměř šestkrát větší než React a dokonce téměř desetkrát objemnější než Vue. I z těchto důvodů se obecně uvádí, že se tento framework používá spíše pro větší projekty [8].

Framework Angular aplikuje znovupoužitelné komponenty s použitím tzv. *two-way data bindingu*, do českého jazyka volně přeloženo jako „oboustranně svázaná data“, jenž řeší synchronizaci dat mezi logickou částí (modelem) a prezentační částí aplikace (*view*). Komponenty jsou navrženy tak, že změny v logické části jsou okamžitě reflektovány i v uživatelském rozhraní a naopak, což vývojářům velmi usnadňuje práci.

React[4]

Javascriptová knihovna React vznikla v roce 2013. Za jejím vznikem stojí společnost Facebook, která ji používá pro vývoj všech svých hlavních produktů (kromě samotného Facebooku také pro vývoj Instagramu a aplikace WhatsApp). React je často mylně představován jako framework, jedná se však o knihovnu rovněž založenou na znovupoužitelných komponentách, jejíž jedinou úlohou je vyrenderování uživatelského rozhraní. Při vývoji komplexnějších aplikací je tak zapotřebí využití knihoven třetích stran, například knihovny `react-router-dom` pro směrování v aplikaci.

React disponuje velmi důkladnou a přehlednou dokumentací. Kromě ní lze na oficiální stránce této knihovny nalézt také tutoriál, který začínajícího uživatele Reactu provede procesem nastavení prostředí včetně základů vývoje v knihovně React. Dle statistik [13]

[2] https://opensource.org/licenses/MIT/
[3] https://angular.io/
[4] https://reactjs.org/

je React považován za nejoblíbenější *frontendový*[5] framework, resp. knihovnu, minimálně v porovnání s javascriptovými frameworky Angular a Vue. Celkového počtu stažení balíčku `react`, se základní funkcionalitou pro tvorbu React komponent, s využitím jednoho z dostupných javascriptových správců balíčků Node.js package manageru[6] (zkráceně npm) bylo v minulém roce 2020 více než 407 milionů. Tedy asi čtyřiapůlkrát více než bylo ve stejném roce stažení základních balíčků pro frameworky Angular i Vue, opět prostřednictvím npm. K 28. prosinci roku 2020 bylo prostřednictvím webové služby GitHub[7] vytvořeno téměř 5,3 milionu repozitářů závislých na balíčku `react`, přibližně třikrát více než pro Vue a Angular. V porovnání s dvěma zmíněnými frameworky se Reactu rovněž týká více dotazů na serveru Stack Overflow, sleduje jej více uživatelů na Twitteru a také poskytuje mnohonásobně více pracovních příležitostí.

Oproti Angularu lze React použít pouze pro čistě front-endové aplikace, především pro vývoj dříve zmíněných *single-page* webových aplikací. React totiž disponuje vlastním virtuálním objektovým modelem dokumentu, což je abstraktní kopie originálního dokumentového objektového modelu (anglicky *Document Object Model*, zkráceně DOM). Změní-li se stav některé z komponent, změní se i příslušná část virtuálního DOMu. Při provádění této změny se stránka nenačítá znovu, nově se vyrenderuje pouze aktualizovaná komponenta. Je vhodné využít této vlastnosti Reactu v dynamických a často se aktualizujících aplikacích. Narozdíl od Angularu, React nativní *two-way data binding* neposkytuje, uživatel si jej však může doimplementovat.

Výhodou ale i nevýhodou knihovny React je, že má více než 1600 přispěvatelů a tak se knihovna neustále vyvíjí. Udržet kód s dokumentací aktuální tak může být složité. React je však vyvíjen se zpětnou kompatibilitou a tak není třeba se obávat, že by mohla webová aplikace s novými vydáními přestat fungovat. Slabší stránkou Reactu je také obtížnější nastavení optimalizace pro vyhledávače, to je však v případě administračního portálu irelevantní.

Vue[8]

Další z populárních javascriptových frameworků, který se na poli *frontendových* technologií objevil až v únoru 2014. O jeho vznik se zasloužil Evan You, jenž se o pár let dříve podílel na vývoji AngularJS.

Mezi Vue a Reactem lze pozorovat spoustu podobností. Vue má totiž také obsáhlou detailní dokumentaci a učící křivka také není tak strmá jako v případě Angularu. Stejně jako React s Angularem, je i Vue založený na znovupoužitelných komponentách a rovněž používá abstraktní kopii DOMu. Ve Vue se jí však neříká virtuální, ale vizuální DOM. Vue se též hodí pro vývoj rychlých dynamických *single-page* webových aplikací a disponuje také vlastním rozšířením vývojářských nástrojů pro prohlížeče.

Narozdíl od Reactu je ale Vue framework. To znamená, že při tvorbě menších aplikací si vystačí sám o sobě a při vývoji těch komplexnějších nepotřebuje tolik závislostí na doplňující knihovny třetích stran. Tento framework dále vyniká svou malou velikostí a také tím, že podobně jako Angular striktně rozděluje logiku od vzhledu, k čemuž využívá již zmíněný *two-way binding*. Pro prezentační část komponent se ve Vue (i v Angularu) pou-

[5] *Frontend* je označení pro prezentační část aplikace, která intereaguje s uživatelem.
[6] https://www.npmjs.com/
[7] https://github.com/
[8] https://vuejs.org/

žívají tzv. *HTML templates* (česky HTML šablony), do kterých jsou dynamicky vkládána data z logické části.

Vue není, narozdíl od zmíněných předchůdců, spravován žádnou velkou společností. Je v rukou, v porovnání s Reactem či Angularem, samostatné malé skupiny přispěvatelů čítající okolo 300 vývojářů. Větší popularity se mu dostává až v posledních letech. S tím souvisí i menší množství rozšiřujících knihoven a balíčků, jichž je většina v čínštině [6].

Kapitola 3

React

Jak bylo zmíněno v kapitole 2, React (také nazývaný jako ReactJS) je knihovna pro tvorbu uživatelských rozhraní. Hodí se pro vývoj rychlých a interaktivních webových aplikací, protože dokáže efektivně aktualizovat části webové stránky bez nutnosti jejího znovunačtení. Především kvůli tomu, a také na základě kritérií zmíněných dříve, byla vybrána tato knihovna pro implementaci administračního portálu pro časosběrný systém. V následujících podkapitolách je představena historie, prostředí pro práci s touto knihovnou a základní principy jejího používání.

3.1 Historie

React poprvé spatřil světlo světa v roce 2011. Vytvořil jej Jordan Walke, softwarový inženýr pracující pro Facebook. Právě tam se React poprvé aplikoval, konkrétně na tzv. *newsfeedu*. O rok později společnost Facebook odkoupila firmu Instagram a začala React používat i tam. V květnu dalšího roku se stal na konferenci javascriptové komunity JSConf 2013 z Reactu *open-source* projekt. React si prakticky okamžitě získal popularitu javacriptové komunity a dnes se řadí mezi nejpoužívanější *frontendové* javascriptové frameworky a knihovny. Dle každoročního průzkumu The State of JS [12] je dokonce nejpoužívanější technologií na poli *frontendu*. Od svého vzniku si React prošel značným vývojem a způsob, kterým se používal v roce 2013 a kterým se používá dnes, je zcela odlišný. To platí i pro jeho syntax, který se měnil společně s tím, jak se měnil JavaScript [2].

3.2 Ekosystém a použité technologie

JavaScript, HTML, CSS

JavaScript je vysokoúrovňový, dynamický, interpretovaný a také objektově orientovaný skriptovací jazyk bez typové kontroly. Od svého vzniku v roce 1995 prošel významným vývojem, který od roku 1997 normuje organizace European Computer Manufacturer Association (ECMA). Podle ní se nazývají jednotlivé standardy – ECMAScript (zkráceně ES), jež JavaScript implementuje. Prozatím poslední vydanou verzí je ta z června loňského roku ECMAScript2020 (ES2020). I zásluhou neziskové organizace ECMA se stal z JavaScriptu jazyk umožňující vývoj *full stack* aplikací[1] [5, 2].

[1] *Full stack* aplikací se rozumí taková aplikace, která vyžaduje vývoj klientské (*frontend*) i serverové části (*backend*).

Základním prvkem každé webové stránky je značkovací jazyk HTML (HyperText Markup Language), který umožňuje definovat strukturu webové stránky. Ta je tvořena HTML elementy, k jejichž zápisu se využívají značky (anglicky *tag*). Jednotlivé elementy pak lze stylovat pomocí kaskádových stylů (Cascading Style Sheet, zkráceně CSS).

Babel[2]

Babel je javascriptový kompilátor, který umožňuje používání prvků nejnovějších verzí JavaScriptu, označovaných jako ECMAScript2015+ či ES6+, a JSX, o kterém bude řeč v následující podkapitole 3.3. Babel kompiluje tyto prvky do zpětně kompatibilních verzí JavaScriptu, které by měly být schopné interpretovat všechny prohlížeče.

Babel vznikl v září 2014 původně pod názvem „6to5“. Jak název napovídá, tento nástroj dokázal konvertovat syntax Javascriptové verze ES6 do verze ES5, který měl tehdy mezi prohlížeči větší podporu. Současný název získal Babel v únoru 2015 [2].

Webpack[3]

Webpack je tzv. balíčkovač (anglicky *module bundler*) pro javascriptové aplikace. Je to nástroj, který umožňuje ze zdrojových souborů různých typů (JavaScript, CSS, Sass, Less, JSX, apod.) vytvářet balíčky. Dle [20] Webpack při zpracování aplikace vytváří graf závislostí (anglicky *dependency graph*), na které se odkazuje z jednotlivých modulů, a generuje jeden či více balíčků.

Hlavní výhody, které používání Webpacku přináší, jsou modularita kódu a rychlost načítání. Balíčkování umožňuje rozdělit zdrojový kód na menší části (moduly), se kterými se snadněji pracuje a lépe se udržují. Používání balíčkování také zvyšuje rychlost načítání, neboť klientovi stačí načíst pouze jeden soubor se všemi závislostmi pomocí jednoho HTTP požadavku (anglicky *request*), což významně snižuje čas načítání celé aplikace. Kromě toho v rámci vytváření balíčku probíhá minifikace kódu, díky které se také snižuje čas načítání [2].

Webpack není jediným dostupným řešením pro balíčkování. Alternativně lze použít také nástroje Browserify[4], Gulp[5], Grunt[6] a další.

npm

Node package manager (zkráceně npm) je, s více než jedním milionem *open-source* balíčků, největším online softwarovým registrem na světě. Kromě toho je npm také nástrojem pro příkazový řádek, díky kterému lze instalovat balíčky a používat je ve vlastních projektech. To vývojářům dává možnost zaměřit se na podstatu problému využitím již dostupných funkcionalit, které nemusí oni sami od začátku implementovat [17].

Obvykle je npm instalován společně s Node.js[7], což je *open-source* běhové prostředí pro javascriptové aplikace, a s npx, nástrojem pro spouštění Node balíčků. Příkladem balíčku, který je spouštěn pomocí npx je Create React App, jímž je inicializován projekt.

Příkazem `npm init`, volaným nad adresářem pro zvolený projekt, se vytvoří soubor `package.json` s inicializační konfigurací:

[2] https://babeljs.io/
[3] https://webpack.js.org/
[4] http://browserify.org/
[5] https://gulpjs.com/
[6] https://gruntjs.com/
[7] https://nodejs.org/en/

```
{
  "name": "test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Příkazem `npm install <název_balíčku>` se do souboru `package.json` do položky `dependencies` (pokud položka neexistuje, vytvoří se) přidá závislost na instalovaný balíček, včetně jeho verze. Samotný balíček se nainstaluje do adresáře `node_modules`.

Node package manager nabízí širší využití, dva uvedené příkazy pouze patří k těm nejpoužívanějším.

3.3 Hlavní koncepty

JSX

Zkratka JSX vychází ze spojení JavaScript (JS) a XML (X). JSX je syntaktické rozšíření JavaScriptu usnadňující tvorbu React elementů, jež se podobá způsobu zápisu HTML. Je totiž také založený na používání *tagů*. Používání JSX zvyšuje čitelnost kódu a efektivitu programování. Definice elementu může vypadat následovně:

```
const element = <h1 className="title">Hello, {name}</h1>
```

Využívání tohoto rozšíření pro tvorbu elementů není povinné. React elementy jdou vytvářet i bez něj. JSX je totiž pouze tzv. *syntactic sugar* pro volání `React.createElement()`. Konečně, do této podoby je JSX syntax díky Babelu kompilován:

```
const element = React.createElement(
    'h1',
    {className: 'title'},
    'Hello',
    {name}
);
```

JSX umožňuje používání javascriptových výrazů. Ty musejí být umístěny do složených závorek, podobně jako dynamicky vložená proměnná `name` v ukázkách výše. React elementy tedy mohou obsahovat například konkatenace řetězců, výpočty, iterace polí s použitím funkce `map()` či podmiňované renderování za použití ternárního operátoru. Při tvorbě elementů a jejich stylovaní je nutné mít na paměti, že slovo `class` je v JavaScriptu rezervované. Specifikovat třídu je však možné pomocí `className` [2].

Znovupoužitelné komponenty

Jak již bylo zmíněno, React je založený na používání komponent. To přináší řadu benefitů. Mezi ně patří možnost dekompozice celého uživatelského rozhraní na drobné nezávislé části, které se snadněji udržují. Tyto drobné části, resp. komponenty, jsou znovupoužitelné. V praxi to znamená, že jednu komponentu lze využít na více místech aplikace. Nevzniká tak duplicitní kód. Je také samozřejmě možné komponentu sdílet i mezi projekty.

V Reactu je možné používat dva typy komponent - třídní a funkční. Třídní komponenta, také označovaná jako stavová (anglicky *stateful*), používá syntax z javascriptové verze ES6 z roku 2015, ve které bylo, poprvé v JavaScriptu, umožněno používání tříd. Při definici třídní komponenty (viz Zdrojový kód 3.1) se uvádí klíčové slovo `class` následované názvem třídy a spojením `extends React.Component`, eventuálně `extends React.PureComponent`. Uvnitř definice třídy pak musí být minimálně metoda `render()`, která vrací JSX. Dále může obsahovat další metody sloužící k práci s životním cyklem komponenty, kterým je věnována podkapitola 3.4. Kromě nich si může programátor nadefinovat i metody vlastní.

```
class HelloWorld extends React.Component {
    render() {
        return (
            <div>
                <h1>Hello World!</h1>
            </div>
        );
    };
}
```

Zdrojový kód 3.1: Třídní komponenta.

Více než třídní, se v dnešní době používají komponenty funkční. Ty byly dříve označované jako bezestavové (anglicky *stateless*), protože neuměly pracovat s vlastním stavem. Uměly pouze přijmout data od svého rodiče a vyrenderovat uživatelské rozhraní. To už dnes neplatí. Vše se změnilo v roce 2019 v rámci nové verze React 16.8, ve které byly představeny tzv. *React Hooks*, o kterých bude řeč v podkapitole 3.5. Díky nim už mohou i funkční komponenty pracovat s vlastním stavem [10].

Definice funkční komponenty vypadá jako klasická javascriptová funkce, která však vrací JSX. V ukázce níže (viz Zdrojový kód 3.2) je použitý nejnovější syntax pro zápis funkce, tzv. *arrow function* syntax.

```
const HelloWorld = () => {
    return (
        <div>
            <h1>Hello World!</h1>
        </div>
    );
}
```

Zdrojový kód 3.2: Funkční komponenta

State versus props

Každá komponenta může mít svůj vlatní stav, který lze v rámci komponenty měnit. V případě třídní komponenty smí být stav jenom jeden, a sice s názvem `state`. Manipulovat s ním lze pouze pomocí metody `setState()`, přímo jej měnit nelze. Tím, že je stav jenom jeden, bývá obvykle reprezentován jako objekt. Ke stavu i jeho metodě se v rámci třídy přistupuje pomocí klíčového slova `this`.

V případě funkčních komponent je možné stav rozdělit do několik proměnných, z nichž má každá i metodu, která její stav aktualizuje. K těmto proměnným se narozdíl od stavu v třídní komponentě přistupuje bez klíčového slova `this`.

`Props` je zkrácená verze slova *properties* (česky vlastnosti). Ty získává komponenta od svého rodiče. `Props` v rámci komponenty nelze měnit. Rodič může předat svému potomkovi nejenom hodnotu či proměnnou, ale také odkaz na metodu, resp. funkci.

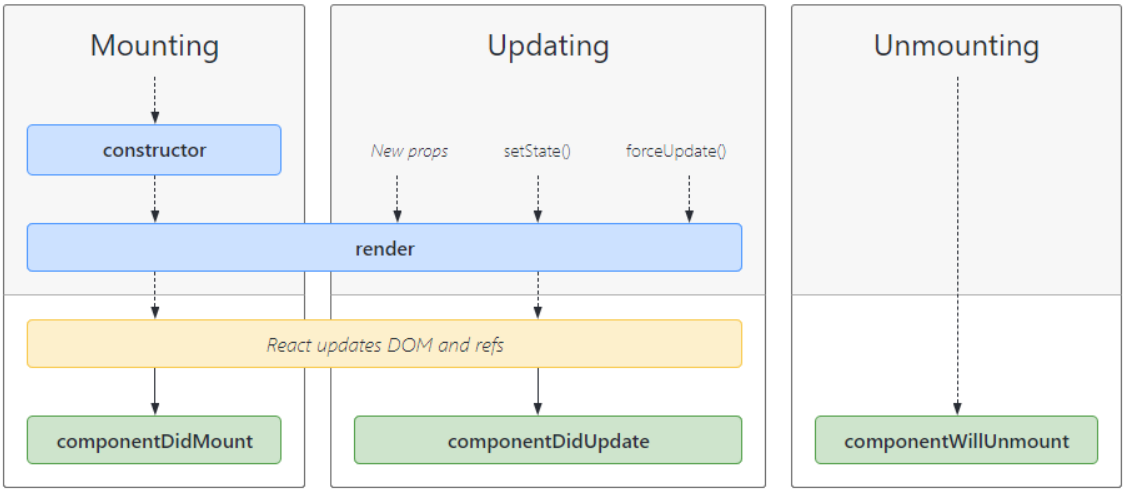
Virtuální DOM

Dokumentový objektový model, zkráceně DOM, je aplikační programové rozhraní, zkráceně API. To reprezentuje XML a HTML dokumenty jako logické struktury ve stromové hierarchii. Každý uzel stromu představuje část dokumentu jako objekt, se kterým lze prostřednictvím DOMu manipulovat. K tomu slouží JavaScript a jeho metody. Například metoda `Document.createElement()`, pomocí které lze vytvořit HTML element, nebo metoda `Node.appendChild()`, díky níž lze vložit element jako posledního potomka příslušného uzlu. Metod pro práci s DOMem existuje spousta, při práci s Reactem se však nepoužívají. React totiž místo tohoto imperativního přístupu umožňuje zaměřit se na to, jak mají vypadat jednotlivé komponenty, jaký stav v nich udržovat, eventuálně jak si jej mezi komponentami sdílet.

Dle [10] je virtuální DOM „programovací koncept, kdy je virtuální reprezentace uživatelského rozhraní uchována v paměti a synchronizována s reálným DOMem pomocí knihovny ReactDOM. Tomuto procesu synchronizace se říká *reconciliation*“ (do češtiny lze *reconciliation* volně přeložit jako smír). V praxi to znamená, že se při změně stavu nebo `props` aktualizuje část virtuálního DOMu, která reprezentuje příslušnou komponentu, případně i její potomky. Jakmile je virtuální DOM aktualizován, provede se synchronizace s reálným prohlížečovým DOMem. Výsledkem toho je re-renderování pouze těch částí uživatelského rozhraní, které byly změněny. Efektivní aktualizace uživatelského rozhraní je tedy ve výchozím stavu plně v režii Reactu a knihovny ReactDOM.

3.4 Životní cyklus komponenty

Každá z třídních komponent, které dávají dohromady uživatelské rozhraní, má svůj vlastní životní cyklus, k jehož fázím se váže několik automaticky volaných metod. Jejich správné použití zvyšuje efektivitu a zároveň snižuje časové prodlevy při načítání webu. Samotný cyklus je rozdělen do tří fází. První z nich se anglicky nazývá *mounting*. Během této fáze je uzel s příslušnou komponentou zařazen do DOMu a vyrenderován. Druhou fází je aktualizace komponenty, která nastává, když se změní její stav, když z rodičovské komponenty dorazí nové `props` nebo když je volána metoda `forceUpdate()`. Poslední fází je tzv. *unmounting*, kdy je uzel s komponentou odebrán z DOMu. Dále jsou představeny nejdůležitější metody pro práci s životním cyklem komponenty, jenž je zobrazen na obrázku 3.1.



Obrázek 3.1: Životní cyklus komponenty, převzato z dokumentace [10].

`constructor()`

Metoda `constructor()` slouží k prvotní inicializaci komponenty, resp. k inicializaci jejího stavu a připojení vytvořených metod. Pokud je však komponenta pouze prezentační[8], metodu `constructor()` ani žádnou jinou nepotřebuje. Tato metoda je volána ještě před tím, než je komponenta přidána do DOMu. Přijímá jako argument ze své rodičovské komponenty `props`. K tomu, aby bylo možné v rámci komponenty k `props` přistupovat, se musí jako první příkaz uvnitř těla metody použít `super(props)`. Dále se inicializuje stav a připojují se vlastně vytvořené metody. `constructor()` pak může vypadat takto:

```
constructor(props) {
  super(props);
  this.state = { show: true };
  this.toggle = this.toggle.bind(this);
}
```

Zdrojový kód 3.3: Metoda `constructor()`.

`componentDidMount()`

V této metodě se provádějí tzv. *side-effects* (česky vedlejší efekty). Ty zahrnují například načítání dat ze serveru zasíláním HTTP požadavků či práci s aplikačním programovým rozhraním prohlížeče. `componentDidMount()` je volána poté, co se vloží komponenta do DOMu a poté, co se vykoná metoda `render()`. Pokud se v rámci této metody mění stav komponenty, je metoda `render()` opětovně vyvolána. Metoda načítající data ze serveru pomocí HTTP klienta `axios` může vypadat takto [10]:

```
componentDidMount() {
    axios.get('https://api.remote_server.com/data')
```

[8]Prezentační komponenta je taková komponenta, která nemá žádný stav (a tedy ani žádné metody pro jeho aktualizaci). Slouží tedy jenom pro zobrazení prvku uživatelského rozhraní.

```
        .then(response => {
            this.setState({ response.data });
        })
        .catch(error => {
            console.log(error);
        })
}
```

Zdrojový kód 3.4: Metoda `componentDidMount()`.

`componentDidUpdate()`

Jak název metody napovídá, je volána poté, co se uskuteční aktualizace komponenty, resp. její opětovné vyrenderování. `componentDidUpdate()` přijímá dva parametry, a sice předchozí stav a předchozí `props`. Ty může ve svém těle porovnávat s novými hodnotami a na základě výsledku porovnání pak například komunikovat se serverem.

`componentWillUnmount()`

Při odebrání komponenty z dokumentového objektového modelu je nutné odstranit vše, co je na ní zavislé a co by mohlo potenciálně zhoršovat výkonnost webové aplikace. Typicky se může jednat například o odstranění *event listeneru*. To by se mělo dít právě v metodě `componentWillUnmount()`.

3.5 React Hooks

Jak bylo uvedeno výše, se vznikem konceptu React Hooks ve verzi 16.8 třídní komponenty přestaly javascriptovým vývojářům dávat smysl. Vše, co dokáží třídní komponenty, od té doby umí i jejich funkční protějšky. Ty jsou však javascriptovým vývojařům bližší díky své syntaxi. Dále je uvedeno několik nejdůležitějších React Hooks, které svými schopnostmi převyšují metody pro práci s životním cyklem třídních komponent.

`useState()`

Chce-li vývojář ve funkční komponentě používat stav, použije k tomu jeden z React Hooks – `useState()`. Jedná se o funkci, která vrací pole o dvou položkách. První z nich je stavová proměnná, druhou je funkce, pomocí které lze stav měnit. Jako parametr se funkci `useState()` předává inicializační hodnota pro stavovou proměnnou. Na příkladu níže je stavová proměnná `counter` inicializovaná na hodnotu 0. Pro její změnu se použije funkce `setCounter()`:

```
const [ counter, setCounter ] = useState(0);
```

`useEffect()`

Druhou hojně používanou funkcí z Hooks je `useEffect()`. Dá se říci, že kombinuje možnosti metod třídních komponent `componentDidMount()` a `componentDidUpdate()`. Již z názvu `useEffect()` je patrné, že také slouží k vykonávání již zmíněných *side-effects*. Podle konfigurace parametrů je funkce vykonána buď pouze po prvním renderu, po každém renderu,

nebo jenom po některých. To je závislé na nastavení volitelného druhého parametru, kterému se říká *dependency array* (česky seznam závislostí). Pokud není nastaven vůbec, `useEffect()` je volaný po každém renderu. Je-li nastaveý jako prázdné pole (resp. seznam), provede se pouze po prvním vyrenderování. Obsahuje-li *dependency array* nějaké proměnné, je funkce volána pouze tehdy, pokud se ve stejnou dobu změní hodnota každé z proměnných v seznamu závislostí. Vykonáním funkce `useEffect()` se rozumí vykonání *callback* funkce, která je funkci `useEffect()` předána jako první parametr. Na ukázce níže je demonstrováno použití, kdy se do konzole zapíše každá změna proměnné `counter`:

```
useEffect(() => {
    console.log("Nová hodnota proměnné counter: ", counter);
}, [ counter ]);
```

`useContext()`

Při používání Reactu je sdílení stavu mezi komponentami běžné. Nastane-li však situace, kdy je v komponentě zapotřebí stav předchůdce, který se nachází o několik úrovní ve stromové hierarchii výše, může se objevit problém. Distribuovat stav komponenty do potomka o několik úrovní níž možné je, ale celá aplikace tím nabývá na složitosti. Řešení poskytuje tzv. *context* a s ním související *context provider* (česky lze volně přeložit jako poskytovatel kontextu). Do něj lze vložit data, která se mají sdílet napříč aplikací nebo pouze její částí. Tu je poté nutné zanořit do *tagu providera* příslušného *contextu*. Hook `useContext()` se používá všude tam, kde je nutné sdílená data využít. *Context* se vytváří následovně:

```
export const DataContext = createContext();

<DataContext.Provider value={{ data }}>
    <App />
<DataContext.Provider>
```

Takto se pak využívá v komponentě, která s daty z *contextu* pracuje (a která se musí nacházet v komponentě `<App />`):

```
const { data } = useContext(DataContext);
```

`useReducer()`

Podle [10] se jedná se o alternativu k funkci `useState()`. `useReducer()` je však vhodnější pro správu komplexní stavové logiky, která zahrnuje tzv. *subhodnoty*, nebo když je zapotřebí odvozovat následující stav od předchozího. V porovnání s `useState()` se kromě iniciální hodnoty stavu předává také jako první parametr funkce, která stav efektivně mění. Definice `useReducer()` vypadá následovně:

```
const [ user, setUser ] = useReducer(
    (user, newUserInfo) => ({...user, ...newUserInfo}),
    initialUser
);
```

`useRef()`

Umožňuje vytvářet reference a přistupovat tak přímo k uzlům umístěným v DOMu. Funkce `useRef()` vrací objekt s položkou `current`. Ta odkazuje na uzel DOMu, kterému je v atributu `ref` příslušného JSX elementu předán název proměnné pro uchování reference:

```
const refElement = useRef();
...
<input ref={refElement} type="text" />
```

Hodnotu elementu `input`, v tomto případě elementu `input`, pak lze získat příkazem `refElement.current.value`. Možností pro práci s uzlem je samozřejmě více.

`useMemo()` a `useCallback()`

Jsou funkce, které používají pro zlepšení výkonnosti techniku zvanou *memoization*. Funkce `useMemo()` používající tuto techniku vrací hodnotu, kterou následně *cachuje* do paměti. Při jejím každém provedení, které se podobně jako u funkce `useEffect` mění na základě *dependency array*, je hodnota uchovaná v paměti porovnávaná s hodnotou novou. V případě rozdílu je použita aktuální hodnota, s jejíž změnou lze dále pracovat. Shodují-li se, je použita již existující reference v paměti. Funkce `useCallback()` funguje na stejné bázi pouze s tím rozdílem, že v paměti přechovává funkci [2].

3.6 Stylování

Specifikovat styly React komponent lze několika způsoby. Prvním z nich je použití inline CSS stylů v HTML elementech. Dále lze definovat vlastní separované CSS soubory, které jsou importovány do souborů s komponentami. Kromě klasických kaskádových stylů lze použít i některý z dostupných CSS preprocesorů, mezi jejichž výhody patří např. možnost vytváření proměnných, zanořování, modularizace stylů, využívání matematických operátorů a další. Významnými CSS preprocesory jsou Sass[9] a Less[10]. Další možností stylování React aplikací je využití Bootstrapu[11], Material UI[12] či jiného dostupného *open-source* CSS frameworku s předpřipravenými komponentami, jejichž styly mohou být upravovány. Výhodou, kterou použití těchto frameworků představuje, je možnost zaměřit se na funkčnost jednotlivých komponent i celé webové aplikace. Využití nastylovaných komponent totiž šetří čas a hodí se tak pro tvorbu aplikací, kde není kladen takový důraz na kreativitu řešení.

3.7 Podpora pro testování

Testování lze rozdělit na automatizované a uživatelské. Nástrojů pro automatizované testování javascriptových aplikací existuje několik. Dle [12] je tím nejpoužívanějším Jest[13]. Při testování React aplikací se společně s ním používá React Testing library[14], případně Enzyme[15].

[9] https://sass-lang.com/
[10] https://lesscss.org/
[11] https://getbootstrap.com/
[12] https://material-ui.com/
[13] https://jestjs.io/
[14] https://testing-library.com/docs/react-testing-library/intro/
[15] https://enzymejs.github.io/enzyme/

Jest je testovací framework pro javascriptové aplikace, o jehož vývoj se stejně jako v případě knihovny React zasloužila společnost Facebook. Výhodou používání Jestu v porovnání s ostatními testovacími frameworky je, že obsahuje všechny potřebné funkcionality pro automatizované testování. Konkrétně se jedná o:

- *assertion* knihovnu umožnující kontrolování správnosti výstupů,
- *test runner*,
- *mocking* a *spying*, metody pro nahrazení reálných funkcí a jejich kontrolu,
- *code coverage report* generující zprávu o pokrytí kódu testy.

Ostatní testovací frameworky (např. Jasmine[16] a Mocha[17]) potřebují ke kompletnímu testovacímu procesu využít jiných nástrojů. Další výhodou Jestu je tzv. *snapshot* testování, které umožňuje ověřovat, zda nedochází k neočekáváným změnám v uživatelském rozhraní. Toto ověřování funguje na základě porovnání výstupu aktuálního testu a *snapshot* souboru s očekávanou strukturou uživatelského rozhraní [9].

Použití React Testing Library v kooperaci s Jestem umožňuje pomocí virtuálního DOMu přistupovat k reálnému DOMu a kontrolovat jeho uzly. To je při testování uživatelského rozhraní React aplikací nezbytné. Využívá se k tomu např. speciální identifikátor HTML elementů `data-testid` či funkce `fireEvent()`, která umožňuje simulovat uživatelovu interakci s webovou aplikací (např. kliknutí na tlačítko nebo vyplnění formulářového pole) [7].

[16] https://jasmine.github.io/
[17] https://mochajs.org/

Kapitola 4

WebSocket

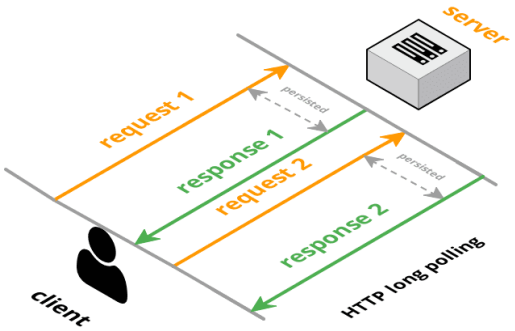
Dle [18] je WebSocket protokol pracující přes HTTP[1], který tvoří perzistentní TCP[2] spojení mezi klientem a serverem. WebSocket umožňuje obousměrnou plně duplexní komunikaci mezi klientem a serverem, což znamená, že mohou oba dva subjekty komunikovat zároveň. To protokol HTTP neumožňuje, byl totiž navržen pouze pro jednosměrnou komunikaci. Výměna informací v případě HTTP probíhá tak, že klient musí nejprve zaslat požadavek na server (HTTP Request), který mu následně odpoví (HTTP Response). Server sám nikdy komunikaci neiniciuje. Alternativou k protokolu WebSocket je technologie *server-sent events*. Ta však v této práci nebyla využita, protože nemá v prohlížečích takovou podporu jako WebSocket.

4.1 Historie

Chtěl-li klient historicky ze serveru získávat data v „reálném čase“, používal k tomu techniku zvanou *polling*, kdy se klient dotazuje na server v pravidelných intervalech. Efektivnější verzí *pollingu* je tzv. *long-polling* (viz schéma na Obrázku 4.1). Tato technika spočívá v tom, že server udržuje spojení po klientově požadavku tak dlouho, jak je možné. Udržuje jej tedy maximálně do doby, než vyprší časový limit pro odpověď. Pak server klientovi odpovídá. Mezi nevýhody tohoto přístupu patří vyšší zátěž serveru či problematické řazení příchozích zpráv ze serveru.

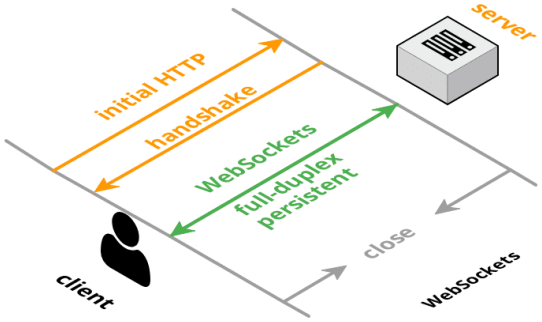
[1] HTTP (HyperText Transfer Protocol) je internetový protokol pro přenos souborů mezi klientem a serverem.

[2] TCP (Transmission Control Protocol) je internetový protokol transportní vrstvy, který zajišťuje spolehlivý datový přenos.



Obrázek 4.1: Komunikační schéma techniky HTTP *long-polling*, převzato z [1].

Proto v roce 2008 začali vývojáři Michael Carter a Ian Hickson přemýšlet nad novým standardem umožňující obousměrnou komunikaci v reálném čase – protokol WebSocket (viz Obrázek 4.2). Již o dva roky později Google Chrome začal protokol WebSocket podporovat. Stal se prvním prohlížečem, který tak učinil. V roce 2011 pak vznikla specifikace RFC[3] 6455, která tuto technologii popisuje. V dnešní době WebSocket podporují všechny hlavní prohlížeče [1].



Obrázek 4.2: Schéma komunikace protokolu WebSocket, převzato z [1].

4.2 Komunikace

Komunikace skrze protokol WebSocket se skládá ze dvou částí – *handshake* a samotný přenos dat. Vytvoření spojení, tzv. handshake, iniciuje klient, který zašle serveru požadavek pro otevření spojení (viz obrázek 4.3). Tento požadavek se nazývá HTTP Upgrade. To kvůli hlavičce `Upgrade` specifikující protokol pro následnou komunikaci a hodnotě `Upgrade` v hlavičce `Connection`, která serveru říká, aby nechal spojení otevřené pro další komunikaci. V hlavičce `Sec-WebSocket-Key` se zasílá náhodně vygenerovaná hodnota, kterou server využije v odpovědi. Další hlavička `Sec-WebSocket-Protocol` definuje subprotokoly na aplikační vrstvě, které klient přijímá. `Sec-WebSocket-Version` poté specifikuje verzi WebSocket protokolu. Verze 13 je jediná validní [11].

[3]RFC (Request for Comments) je označení pro dokumenty popisující internetové protokoly, výzkumy, metody, systémy apod.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Obrázek 4.3: HTTP Upgrade od klienta, převzato ze standardu [11].

Server odpovídá zprávou (viz obrázek 4.4) s kódem `HTTP 101 Switching Protocols`, kterou informuje klienta, že se přepíná do protokolu, jenž klient požadoval. V hlavičce `Sec-WebSocket-Accept` zasílá již zmíněnou náhodně vygenerovanou hodnotu, která je konkatenována s GUID, *hashována* algoritmem SHA-1 a kódována algoritmem base64. Tím je ustaveno spojení, které mohou uzavřít jak server, tak i klient zasláním zprávy s operačním kódem `0x8`.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Obrázek 4.4: HTTP Upgrade od serveru, převzato ze standardu [11].

Kapitola 5

Návrh řešení

5.1 Popis systému a požadavky na navrhované řešení

Jak bylo zmíněno již na úvod, brněnská firma BIXION se zabývá vývojem časosběrných systémů. Jejím hlavním produktem je BixiCon intervalometr (viz Obrázek 5.1), což je zařízení, které slouží k automatickému pořizování snímků v určitých intervalech s řadou několika dalších funkcí, které budou dále specifikovány. Zákazník, který si zakoupí zmíněný intervalometr (dále už jen jednotka), získá kromě fyzického řešení také přístup do administračního portálu[1] a na jeden měsíc předplacenou službu vzdálené správy pro danou jednotku. Pro vzdálenou správu jednotky má firma BIXION samostatnou webovou aplikaci, která není předmětem této práce. Prostřednictvím admininistračního portálu by měl mít uživatel kontrolu nad všemi svými jednotkami a právě z portálu pro administraci by měl mít možnost přistoupit do aplikace pro vzdálenou správu konkrétní jednotky.

Obrázek 5.1: BixiCon intervalometr, převzato z webové stránky [3].

[1] https://is.bixion.com

K tomu, aby si mohl uživatel prohlížet pořízené fotky a z nich vygenerovaná videa, slouží další placená služba – Prohlížeč. Zakoupením jednoho prohlížeče získá uživatel možnost připojit k němu své jednotky a online tak prohlížet pořízené snímky a videa z připojených jednotek v separované aplikaci[2]. Dále uživatel disponuje diskovým prostorem na cloudovém úložišti, ke kterému má rovněž přístup.

Přehled jednotek by měl obsahovat pouze nejpodstatnější informace. Jmenovitě mezi ně, kromě názvu a vlastníka, patří také mód a stav, ve kterém se jednotka právě nachází, přičemž jednotky rozlišují dva módy. Úsporný (anglicky *powersave*) mód, jenž může nabývat stavu online, offline a režimu spánku, a normální mód, který může být buď ve stavu online, nebo ve stavu offline. Dále by mělo být z přehledu zřejmé, kdy se naposled jednotka aktualizovala a jaký je na ní aktuální čas, počet vyfocených a nahraných snímků a jejich chybovost, provozuschopnost a v neposlední řadě také napětí a teplota jednotky. Detailnější informace o jednotkách včetně jejich konfigurace nalezne uživatel v samostné aplikaci pro vzdálenou správu jednotek. Uživatel by měl mít možnost se do této aplikace dostat přímo z administračního portálu.

Portál pro administraci časosběrného systému pochopitelně musí disponovat možností konfigurace osobních údajů, včetně hesla a daňového identifikačního čísla (DIČ), které se používá při vystavování faktur za již zmíněné služby. Konfigurovat lze i jednotlivé prohlížeče a to konkrétně jejich název, URL adresu, popis, logo, čas snímků, možnost generování videa a rozmazání obličejů, barevné schéma a jednotky k prohlížení.

Protože zákazníci firmy BIXION nepochází pouze z tuzemska, je zapotřebí, aby byl administrační portál dostupný kromě českého jazyka minimálně také v jazyce anglickém. Další funkcionalitou, kterou by mohl portál pro administraci disponovat, je napojení na platební bránu, aby měl uživatel možnost prodloužit platnost zakoupených služeb.

Konfigurovatelné pohledy na aplikaci a jejich omezení

Administrační portál musí rozlišovat 3 typy pohledů na aplikaci – administrátor, distribující administrátor a zákazník. Výše zmíněný popis obsahuje vše, co bude mít dostupné v administračním portálu uživatel s přístupovými právy zákazníka s tím, že si pochopitelně bude moci zobrazit a pracovat pouze s jednotkami a prohlížeči, které sám vlastní.

Oproti tomu distribující administrátor (anglicky *reseller admin*) bude mít kromě svých jednotek a prohlížečů k dispozici také přehled o jednotkách a prohlížečích všech svých zákazníků. Kromě toho bude moci zákazníkům vytvářet, odstraňovat a spravovat účty.

Administrátor, jako uživatel s nejvyššími právy, musí mít možnost spravovat všechny uživatele, všechny jednotky a všechny prohlížeče.

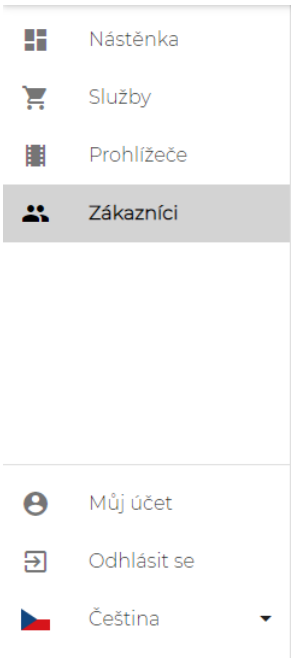
5.2 Návrh uživatelského rozhraní

K návrhu vlastního řešení byl využit online nástroj Figma[3], na jehož základě byla vytvořena výsledná webová aplikace. Výběr základních barev se odvíjel od barevného schématu webových stránek a od loga společnosti BIXION. Jako primární barva tedy byla zvolena sytě světlá zelená, doplňkovou sekundární barvou je šedá.

Administrační portál byl rozdělen do čtyř logických celků – **Nástěnka** (anglicky *Dashboard*), **Služby** (anglicky *Services*), **Prohlížeče** (anglicky *Browsers*) a **Zákazníci** (anglicky *Customers*). Přesouvání mezi těmito sekcemi poskytuje postranní navigace, tzv. *drawer*

[2] https://time.bixion.com
[3] https://figma.com/

zobrazený na obrázku 5.2, odkud se lze dostat také do nastavení vlastního profilu. Drawer uživateli nabízí také možnost odhlášení pomocí jednoho kliknutí a také uživateli umožňuje zvolit jazykovou verzi administračního portálu.



Obrázek 5.2: Návrh postranní navigace.

Sekce Nástěnka

V této sekci má uživatel přehled o všech jemu dostupných jednotkách. Může si zvolit, zda si je přeje zobrazit ve výchozím mřížkovém režimu nebo v režimu řádkovém. Ikona, která poskytuje možnost přepínání mezi těmito režimy, se nachází vpravo od postranní navigace nad jednotlivými jednotkami. Na stejné úrovni v pravé části okna je umístěno textové formulářové pole pro možnost vyhledávání jednotek podle názvu, vlastníka či sériového čísla. Tato funkce poslouží především uživatelům s přístupovými právy administrátora a dále distribujícího administrátora, jimž se může na nástěnce naakumulovat větší množství jednotek.

V záhlaví boxu jednotky lze kromě názvu a vlastníka, a všech dalších uživatelů připojených k jednotce, najít také informaci o módu a stavu jednotky. Písmeno „P“ symbolizuje úsporný (anglicky *powersave*) mód a písmeno „N“ mód normální. Zelená, červená a oranžová barva pak značí stav jednotky, kde oranžová signalizuje uspanou jednotku. V hlavní části boxu jednotky jsou umístěny hlavní informace o jednotce, jež jsou jmenovány v 5.1. Po přemístění kurzoru myši nad některé z polí tabulky se zobrazí nápověda s významem příslušné buňky. Pod tabulkou se nachází prozatím nefunkční tlačítko pro službu online monitoringu, která bude společností BIXION implementována v budoucnosti, a tlačítko „Připojit“, které uživatele přesměruje do vzdálené správy příslušné jednotky.

Obrázek 5.3: Návrh nástěnky s uživatelovými jednotkami v mřížkovém režimu zobrazení.

Sekce Služby

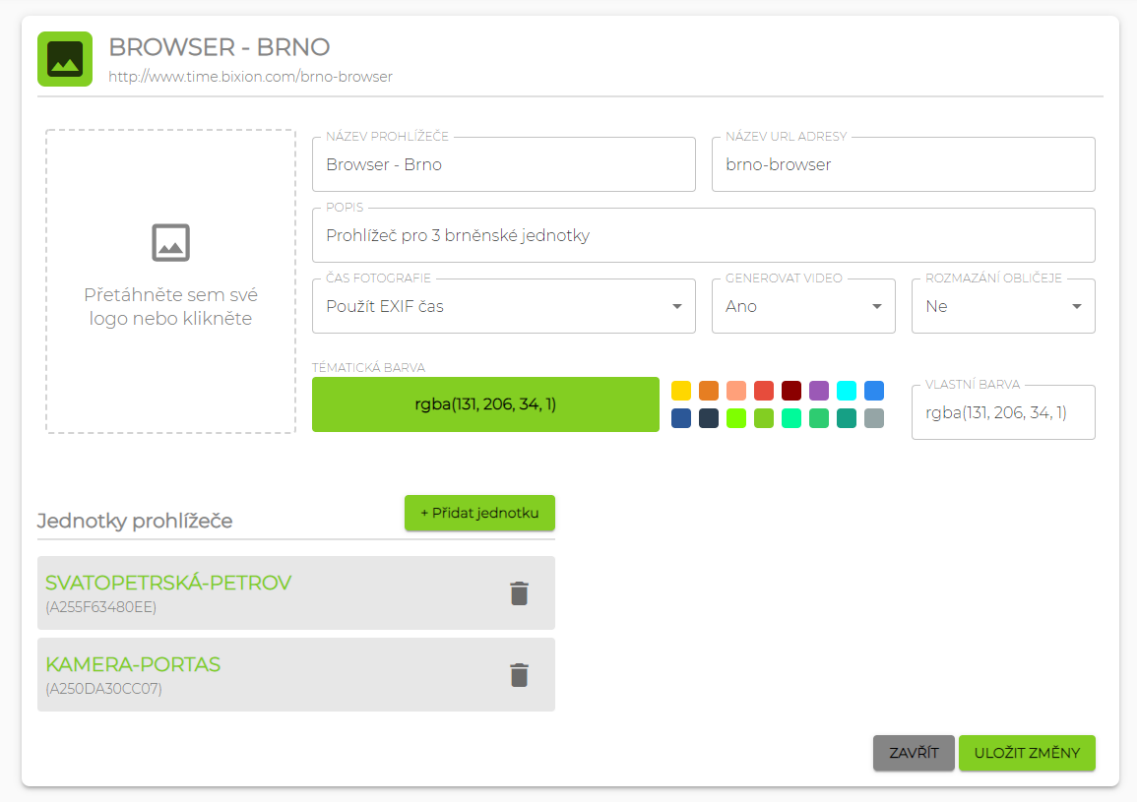
Zde si může uživatel prodloužit platnost služby vzdálené správy, a v budoucnu i služby online monitoringu, svých jednotek a služby prohlížečů včetně navýšení jejich datového prostoru, které je také plánováno výhledově. Po vstupu do této sekce se uživateli zobrazí jednoduchý rozcestník, jenž mu umožní zvolit nákup služeb pro jednotky či pro prohlížeče. V obou případech se zákazníkovi zobrazí tabulka všech jeho jednotek, resp. prohlížečů, a dostupných služeb. Uživatel má v každém řádku u každé jednotky či prohlížeče informaci o tom, dokdy je příslušná služba aktivní. V případě prohlížeče bude také informován o tom, jak velkým datovým prostorem prohlížeč disponuje a z jak velké části je tento prostor zaplněný. Informaci o prodloužení či navýšení může uživatel pomocí čísla vložit přímo do textového pole nebo k tomu může využít přidružená tlačítka „+“ a „-“, pro inkrementaci, resp. dekrementaci, příslušného číselného údaje. Samozřejmostí je možnost výběru několika, případně všech, řádků tabulky, a tedy zadání stejných hodnot pro vícero jednotek nebo prohlížečů. Je-li uživatel se zadáváním hodnot hotový, pomocí tlačítka ve spodní části sekce „Přejít k sumarizaci“ přejde ke shrnutí celé objednávky. Zde si uživatel ověří všechny objednávané položky a bude moci zvolit preferovanou platební metodu – digitální peněženka PayPal nebo kreditní karta, a stisknutím tlačítka „Přejít k pokladně“ je přesměrován.

Obrázek 5.4: Návrh sekce Služby.

Sekce Prohlížeče

Jak název napovídá, sekce Prohlížeče slouží jako přehled jednotlivých prohlížečů. V levém horním rohu této scény má uživatel možnost, po kliknutí na příslušné tlačítko, vytvořit nový prohlížeč včetně jeho počáteční konfigurace. Podobně jako v sekci Nástěnka může uživatel vyhledávat prohlížeče podle názvu.

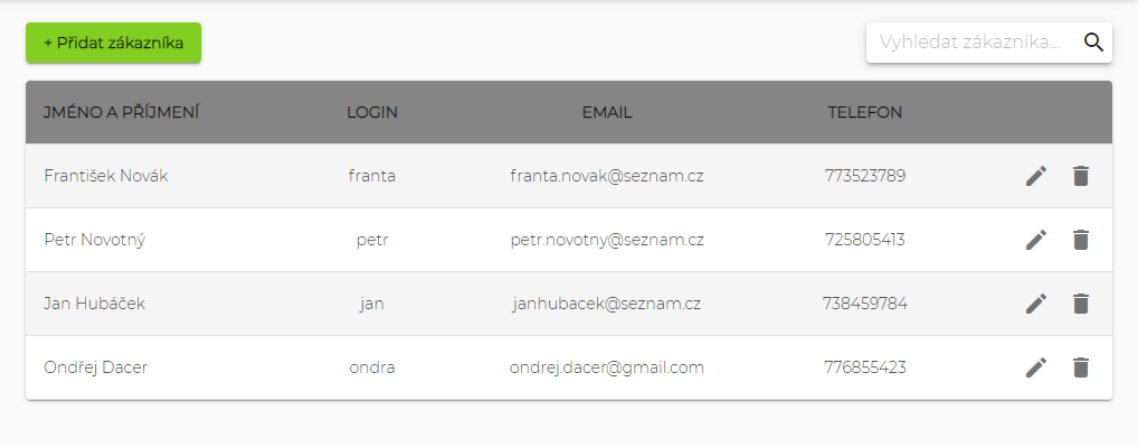
Box prohlížeče obsahuje pouze jeho název a poslední pořízenou fotku z připojených jednotek jako pozadí. V pravé části boxu se nachází ikona pro konfiguraci a pod ní ikona odkazující do samostatné aplikace pro prohlížení snímků a vygenerovaných videí. Detail konfigurace prohlížeče je na obrázku 5.5.



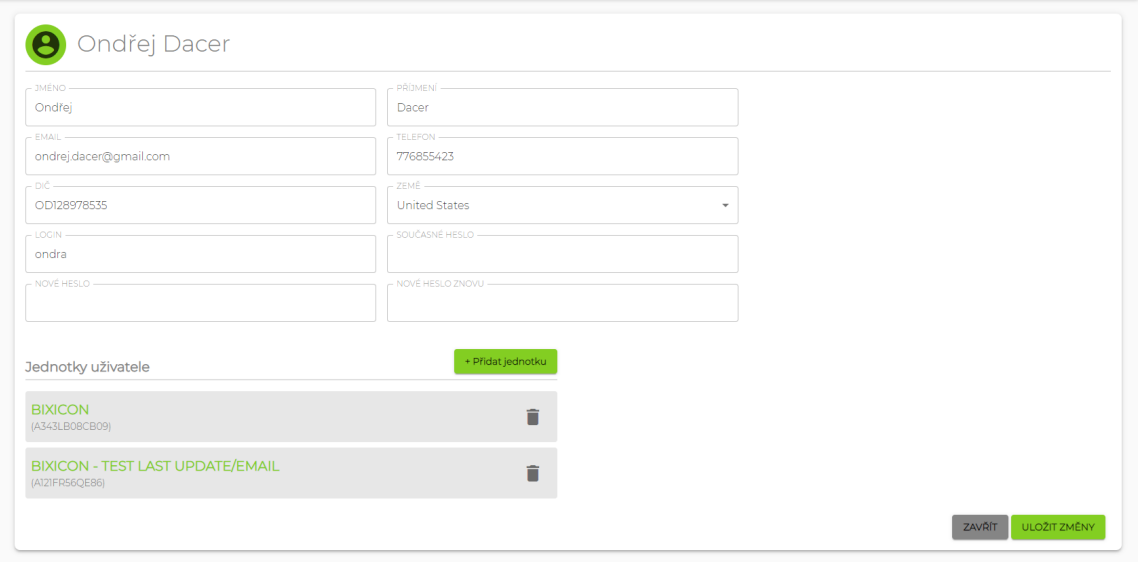
Obrázek 5.5: Konfigurace prohlížeče.

Sekce Zákazníci

K poslední sekci budou mít přístup pouze oba typy administrátorů. Slouží totiž ke správě zákazníků – vytváření nových účtů, jejich správa a odstraňování již neaktuálních či neaktivních uživatelských účtů. I v této sekci bude možné vyhledávat zákazníky, konkrétně podle příjmení a přihlašovacího jména. Přehled zákazníků je zobrazen pomocí jednoduché tabulky se základními údaji o uživateli 5.6.



Obrázek 5.6: Návrh sekce Zákazníci.



Obrázek 5.7: Konfigurace uživatele.

Kapitola 6

Implementace

V této kapitole jsou popsány nejstěžejnější části implementace administračního portálu pro časosběrný systém. Prerekvizitou pro správné spuštění a práci s aplikací vyvíjenou v Reactu je nainstalovaný Node.js společně s npm a npx zmíněnými v kapitole 3.

6.1 Založení projektu

K založení projektu byl využit nástroj Create React App. Spuštěním příkazu `npx create-react-app <název_projektu>` se vytvoří adresář s názvem projektu, uvnitř kterého je vytvořena základní React aplikace. Ta obsahuje tři adresáře:

- `node_modules` obsahující nainstalované závislosti,
- `public` se souborem `index.html`, se souborem `manifest.json` se základní charakteristikou webu, dále se souborem `robots.txt` pro robotické prohledávače webových stránek a s ikonami webu,
- `src` se zdrojovými soubory.

V režii nástroje Create React App je také konfigurace Webpacku, společně s Babelem a instalace základních balíčků pro práci s Reactem, Kromě dříve zmíněných balíčků `react` a `react-dom` se jedná i o balíček `react-scripts`, díky kterému lze spouštět nad aplikací skripty pro spuštění aplikace, její sestavení, testování či pro změnu konfigurace. Dále jsou instalovány testovací balík `@testing-library` a balík `web-vitals` pro sledování metrik.

Nejprve bylo nutné si vytvořit vlastní adresářovou strukturu na základě dekompozice navrhovaného řešení. Každá stránka webové aplikace disponuje vlastní komponentou ve vlastním souboru, který je umístěný v adresáři `Scenes`. Komponenty, ze kterých jsou stránky skládány, se nachází v adresáři `Components`. Složka `Config` obsahuje konfigurační soubory. V adresáři `Containers` se nachází tzv. *reducery*, což jsou funkce pro úpravu stavů aplikace, které jsou definovány v adresáři `Context`. Adresář `Locales` obsahuje jazykové verze webu. Pomocné funkce se nachází v adresáři `Utils`.

Není-li změněna konfigurace Webpacku, je hlavním souborem `index.js`, nacházející se v kořenovém adresáři zdrojových souborů. Zde se totiž do webové stránky, jejíž struktura je definovaná v souboru `index.html` v adresáři `public`, vkládá komponenta s celou React aplikací, konkrétně do *tagu* `div` s vlastním unikátním identifikátorem. K vložení aplikace do konkrétního uzlu se používá metoda `render()` z balíčku `react-dom`:

```
ReactDOM.render(<App />, document.getElementById('root'));
```

K tomu aby bylo možné nad celou aplikací udržovat stav je nutné ji zanořit do *provideru*, resp. do jeho *tagu*. Jedním z nich v administračním portálu je `I18nextProvider` importovaný z knihovny `react-i18next` a s ním související vlastně vytvořený `LanguageProvider`.

6.2 Jazykové verze

Oba tito *provideři* se podílejí na tom, aby bylo možné přepínat mezi jazykovými verzemi. Komponenta `I18nextProvider` přijímá *property* `i18n`, jíž se předává reference na soubor s rozhraním `i18n` importovaným z knihovny `i18next`. V něm jsou definovány cesty k souborům obsahující anglické a české jazykové verze a způsob, jakým jsou oddělovány jednotlivé překlady. `LanguageProvider` pak umožňuje přistupovat k aktuálně nastavenému jazyku a měnit jej. Překládat lze pomocí funkce `t()`, která je součástí *hooku* `useTranslation()` z knihovny `react-i18next`. Využití překladu vypadá takto:

```
<Typography variant="subtitle1" className={classes.tableHeadCellTitle}>
    {t('services.browsersTable.title.browserName')}
</Typography>
```

6.3 Material UI

K tvorbě webové aplikace byl využit framework pro tvorbu uživatelského rozhraní Material UI, který byl zmíněn v kapitole 3 a jehož balíčky `@material-ui/core` a `@material-ui/icons` lze nainstalovat pomocí npm. V souboru obvykle pojmenovaném `theme.js` je možné definovat některé základní styly (např. barvy či *breakpointy*) a uložit je do proměnných, které lze následně používat napříč celou aplikací. K tomu je zapotřebí „obalit“ celou aplikaci Material UI komponentou `MuiThemeProvider`, která přijímá *property* `theme`, které je předán právě soubor se základními styly. Veškeré stylování se provádí v rámci komponent, k čemuž je využívána funkce z Material UI `makeStyles()`, která využívá jako parametr zmíněný `theme`. Definice stylů pro komponentu vypadá následovně:

```
const useStyles = makeStyles((theme) => ({
    button: {
        fontWeight: 600,
        color: 'black',
        height: 'fit-content',
        [theme.breakpoints.down('xs')]: {
            fontSize: '12px'
        },
    }
}));
```

6.4 Směrování a knihovna `react-router-dom`

Celá aplikace je zanořena také do třídy `BrowserRouter` využívající objekt HTML5 `history`, pomocí kterého lze snadno pracovat s URL. Využití této třídy umožňuje směrování v rámci aplikace. V `<App />` se obvykle používá třídní komponenta `Switch` s potomky třídy `Route`,

které symbolizují cesty (anglicky *path*) pro jednotlivé stránky. `Switch` postupně prohledává všechny `Route` a jakmile narazí na shodu v cestě, zobrazí příslušnou komponentu.

V případě administračního portálu to ale nestačí. Implementovaná webová aplikace disponuje stránkami, které jsou dostupné pouze přihlášenému uživateli. Volně přístupná je pouze stránka pro přihlášení. Za tímto účelem byly vytvořeny vlastní komponenty `PrivateRoute` (viz Zdrojový kód 6.1) a `ProtectedLogin`. Informaci, zda je uživatel autentizován, lze ve webové aplikaci zjistit z hodnoty proměnné `auth` definované v `AuthApiContext`. Nabývá-li *booleovské* hodnoty `true`, znamená to, že je uživatel přihlášen a že se má zobrazit komponenta pro danou cestu. Komponenty pro autentizované uživatele jsou vnořené do komponenty `Layout`, v rámci které je zobrazena horní lišta a postranní navigace. Pokud je zadána cesta, která není definovaná, je uživatel přesměrován na stránku pro přihlášení v případě, že není přihlášen, a na stránku **Nástěnka** v opačném případě. Přesměrování zajišťuje komponenta `Redirect` z knihovny `react-router-dom`, ze které jsou rovněž komponenty `BrowserRouter, Switch, Route`.

```
const PrivateRoute = ({ auth, component:Component, ...rest }) => {
    return(
        <Route
            {...rest}
            render = {(props) => auth
            ?
                <Layout>
                    <Component {...props} {...rest} />
                </Layout>
            :
                <Redirect to="/login" />}
        />
    )
}
export default withRouter(PrivateRoute);
```

Zdrojový kód 6.1: Komponenta `PrivateRoute`.

6.5 Aplikační programové rozhraní a knihovna `axios`

Pro dotazování se na server, resp. jeho REST[1] API, je primárně využívána vlastně vytvořená funkce `BackendRequest`, která implementuje knihovnu `axios`. Ta umožňuje tvorbu HTTP požadavků a následné zpracování HTTP odpovědí za použití tzv. *Promise*. Funkce jako parametry, které jsou používány pro vytvoření požadavků, přijímá:

- metodu požadavku,
- adresu dotazovaného *endpointu*,
- data (pouze pro metodu POST),
- *callback* funkci, která je vykonána, vrátí-li server validní odpověď,

[1] REST je zkratka pro *Representational State Transfer*. Jedná se o architekturu rozhraní umožňující práci s daty pomocí protokolu HTTP.

- *callback* funkci, která je vykonána, nastane-li chyba.

Kromě uvedených parametrů jsou při tvorbě HTTP požadavků specifikovány hlavičky `Content-Type` a `Accept` s hodnotou `application/json`. Těmi je serveru sděleno, že klient zasílá a přijímá data ve formátu JSON. Dále je ale především definována hlavička `Authorization` s hodnotou `Bearer` následovanou tokenem daného uživatele, který generuje server při přihlášení do administračního portálu. Autentizační token je uchováván v paměti prohlížeče, konkrétně v tzv. `localStorage`. Tímto tokenem se uživatel prokazuje při každém HTTP požadavku na server. Za účelem práce s `localStorage` byly vytvořeny dvě pomocné funkce: `setToken()` a `getTokenFromLocalStorage()`.

Ve většině případů je server dotazován prostřednictvím uvedené funkce `BackendRequest`. V případě tzv. *custom hooku* `useBrowserForm()` zpracovávající formuláře pro tvorbu a úpravu prohlížečů je však využit HTTP klient `axios` přímo, a sice pro nahrání prohlížečového loga. Děje se tak z důvodu odlišných hlaviček požadavku (pro nahrání souboru `Content-Type: 'multipart/form-data'`) a také kvůli tomu, aby byla zajištěna asynchronicita. Logo totiž musí být nahráno dříve, než je vytvořen či upraven prohlížeč. Asynchronicita tohoto serverového dotazu je zajištěna klíčovými slovy `async` v definici funkce a `await` v jejím těle před voláním HTTP klienta.

Custom hooky `useUnitsSearch()`, `useBrowsersSearch()` a `useCustomersSearch()` sloužící ke stránkování a vyhledávání rovněž využívají `axios` napřímo. Zde kvůli tomu, že je potřeba pracovat s tzv. *cancel tokenem*, což je reference na příslušný požadavek. Zadává-li uživatel do pole pro vyhledávání řetězec, s každým jedním znakem je vyvolán požadavek na server. Pokud znaky zadává uživatel v rychlém sledu po sobě, může se stát, že je serveru zaslán požadavek ještě předtím než dorazí odpověď na požadavek předchozí, což může vést k problémům s nekonzistencí dat. Řešením je *cancel token*, jenž umožňuje uzavřít poslední HTTP požadavek.

Stránkování a vyhledávání

Jedním z požadavků na portál pro administraci byla možnost vyhledávání a stránkování jednotek, prohlížečů i zákazníků. Pokud by se mělo vše načítat najednou, mohlo by to trvat příliš dlouho, nebo by se případně nemuselo načíst vůbec nic kvůli vypršení časového limitu pro HTTP odpověď. Za tímto účelem byly na straně serveru vytvořeny *endpointy* se třemi parametry – číslo stránky, její velikost a vyhledávaný řetězec. Příslušný *endpoint* je volán z jednoho z *custom hooků* zmíněných výše. Tomu je z komponenty pro danou stránku předáváno číslo stránky a vyhledávaný řetězec.

Stránky **Nástěnka**, **Prohlížeče** a **Zákazníci** implementují tzv. *infinite scroll*. Jedná se o techniku, která umožňuje načítat obsah po částech. K tomu je zapotřebí tzv. *observer*, což je element stránky, který se nachází pod aktuálně načteným obsahem. Jakmile se uživatel posouvá webovou stránkou směrem dolů a narazí na něj (*observer* se bude nacházet ve viditelné části stránky), načte se další obsah. V případě implementované webové aplikace se načtou další data ze serveru.

Je-li dosaženo *observeru*, který v implementované webové aplikaci představuje *spinner*, je inkrementováno číslo stránky. To v příslušném *custom hooku* vyvolá požadavek na server na další stránku. Komponentě se stránkou jsou zpět vráceny:

- informace, zda se ještě načítají data,
- informace, jestli server disponuje dalšími daty,

- informace o potenciální chybě, která nastala,
- všechny vrácené položky ze serveru uložené v poli.

Zadáním řetězce do pole pro vyhledávání se hodnoty všech ostatních proměnných změní na původní hodnoty.

6.6 Contexty a práce s daty

`BackendDataContext`

Všechna příchozí data ze serveru jsou uchovávána v rámci `BackendDataContext`. Konkrétně se jedná o data o jednotkách, prohlížečích, zákaznících a o přihlášeném uživateli, která se ukládají do následující struktury:

```
const BACKEND_DATA_STRUCTURE = {
    user: {},
    units: {},
    browsers: {},
    customers: {},
}
```

K aktualizaci stavu se využívá funkce `dispatch()`, která přijímá jako parametr tzv. akci. Jedná se o objekt minimálně s položkou `type` specifikující příslušnou akci. Další položky jsou závislé na typu akce. Veškeré akce jsou definovány v *reduceru* `backendDataReducer` v rámci konstrukce `switch`. Jedná se o:

- `FETCH_STRUCTURE` – inicializace základní struktury,
- `FETCH_USER` – uložení konfigurace aktuálně přihlášeného uživatele,
- `UPDATE_USER` – její aktualizace,
- `FETCH_UNITS` – uložení načtených jednotek,
- `UPDATE_UNIT` – aktualizace jedné konkrétní jednotky (buď přepnutím *switche* u jednotek v úsporném (*powersave*) módu, nebo na základě zprávy z WebSocketu, viz podkapitola 6.7),
- `FETCH_BROWSERS` – uložení načtených prohlížečů,
- `FETCH_CUSTOMERS` – uložení načtených zákazníků.

Data o prohlížečích a zákaznících nejsou ukládána za účelem jejich aktualizace, ale kvůli tomu aby nemusela být načítána znou v případě, že uživatel přejde na stránku konfigurace prohlížeče, resp. zákazníka.

`OrderContext`

Dalším *contextem* pracujícím s daty je `OrderContext`, který shromažďuje data o objednávaných položkách. I zde je pro úpravu dat použit *reducer* (s akcemi `UPDATE_ORDER` pro úpravu a `DELETE_ORDER` pro zrušení objednávky), neboť je práce se stavem objednávek komplexnějšího charakteru. Stránka **Služby** je rozdělena na dvě podstránky a stejně tak i struktura pro stav objednávek:

```
const ORDER_STRUCTURE = {
    units: {
        price: 0,
        order: []
    },
    browsers: {
        price: 0,
        order: []
    },
}
```

V původním návrhu uživatelského rozhraní bylo uvažováno veškeré objednávání na jedné stránce. V případě velkého množství jednotek a prohlížečů by ale navržené řešení postrádalo přehlednost. Kvůli tomu se nyní objednávají jednotky a prohlížeče zvlášť.

`ACLContext`

Administrační portál využívají uživatelé s různými přístupovými právy a některé části administračního portálu jsou tak dostupné na základě uživatelovy role. Typ uživatele lze zjistit z dotazu na *endpoint* `/user`. Podle něj je nastavena stavová proměnná `permission` komponenty `ACLContext`, jejíž hodnotou je pole. To je tvořeno prvky reprezentující jednotlivé role. Je-li uživatel adminem, obsahuje stavová proměnná všechny existující role: `['ACL_ADMIN', 'ACL_RESELLER_ADMIN', 'ACL_CUSTOMER']` apod.

Společně s tímto *contextem* byla vytvořena i komponenta `ACLWrapper`. Do ní je zanořen obsah, který má být viditelný pouze pro administrátora, případně i pro *reseller* admina. Hierarchicky nejnižší možná role pro zobrazení obsahu v komponentě `ACLWrapper` jí je předána skrze *property* `requiredPermission`. Komponenta `<UnavailableForCustomers />`, na ukázce níže, je dostupná pro oba typy administrátorů:

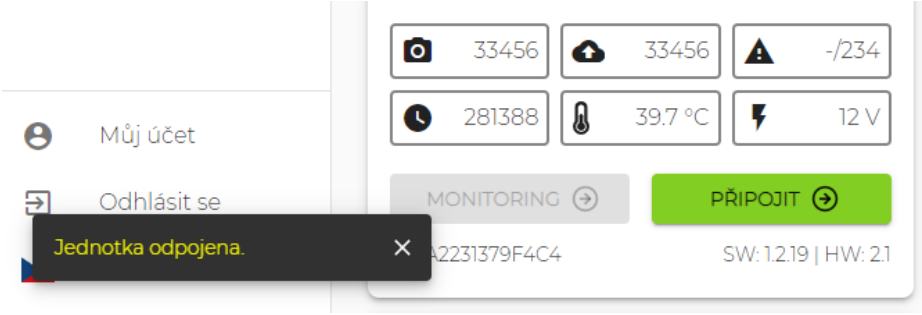
```
<ACLWrapper requiredPermission={'ACL_RESELLER_ADMIN'}>
    <UnavailableForCustomers />
</ACLWrapper>
```

`SnackbarContext`

Název `SnackbarContext` vychází z komponenty `Snackbar` z Material UI, která slouží k zobrazování notifikací (viz obrázek 6.1) ve webové aplikaci. Je také známa pod názvem *toast*. Administrační portál zobrazuje tři typy notifikací – `info`, `warning` a `error`, které jsou barevně odlišené a které jsou zobrazeny v levém dolním rohu obrazovky po dobu čtyř vteřin. Vytvořit notifikaci lze pomocí volání funkce `setSnack()` ze `SnackbarContext`:

```
setSnack({
```

```
        message: t('snackbar.message.changesSaved'),
        type: "info",
        open: true
    });
```



Obrázek 6.1: Ukázka notifikace.

`DashboardDisplayContext` a `DrawerContext`

Poslední dva vytvořené *contexty* umožňují přepínání mezi mřížkovým a řádkovým režimem na stránce **Nástěnka** a otevírání/zavírání navigace na mobilních zařízeních.

6.7 Integrace technologie WebSocket

Je běžné, že se data o jednotkách a prohlížečích, případně i o zákaznících, mění během časového úseku, kdy uživatel pracuje s administračním portálem. Tyto změny ale nebyly reflektovány, vždy bylo zapotřebí aktualizovat webovou stránku. Použitím protokolu WebSocket lze aktualizovat data v reálném čase. Za tímto účelem byl vytvořen `WebSocketContext`, pomocí jehož funkce `setSocket()` je v komponentě `<App />` ustanovováno spojení se serverem. K tomu se využívá funkce `io()` z knihovny `socket.io-client`, které je předávána URL adresa serveru a speciální hlavička `token` s tokenem přislušného uživatele:

```
setSocket(
    io('https://api.bixion.com:8778', {
        extraHeaders: {
            "token": getTokenFromLocalStorage()
        }
    })
)
```

Zprávy z vytvořeného komunikačního kanálu jsou zpracovávány v samotném *contextu*. Aktuální řešení umožňuje aktualizaci dat z jednotek a zobrazení notifikací ze serveru. V budoucnu může být využití protokolu WebSocket rozšířeno i pro *real-time* aktualizaci dat prohlížeče. Zpracování zprávy pro aktualizaci jednotky (typ zprávy `'dashboard_update'`) vypadá následovně:

```
socket.on('dashboard_update', (data) => {
    dispatch({
```

```
            type: UPDATE_UNIT,
            serial: Object.keys(data)[0],
            data: Object.values(data)[0]
        })
    });
```

6.8 Integrace digitální peněženky PayPal

Aby bylo možné provádět v administračním portálu platby, byl integrován jeden z dostupných platebních systémů, konkrétně digitální peněženka PayPal. Dále je pro integraci připraveno také napojení na platební bránu Stripe[2]. Díky tomu bylo možné systém rozšířit v sekci **Služby** o možnost prodloužení služby vzdálené správy pro jednotky a služby prohlížeče až na 99 měsíců dopředu. Integrace digitální peněženky byla umožněna díky využití knihovny `react-paypal-express-checkout` a její komponenty `<PaypalExpressBtn />`:

```
<PaypalExpressBtn
    env={env} // development/produkce
    client={client} // ID webové aplikace
    currency={currency} // měna
    total={total} // celková cena
    onError={onPaymentError}
    onSuccess={onPaymentSuccess}
    onCancel={onPaymentCancel}
/>
```

[2] https://stripe.com/en-cz

Kapitola 7

Testování

V prvé řadě bylo nutné si ujasnit, co je nutné otestovat automatizovaně a co je výhodnější otestovat uživatelsky. K provedení jednotkových, *snapshot* a integračních testů byly využity dostupné nástroje zmíněné v kapitole 3, Jest a React Testing Library. Uživatelské testování vycházelo z předem definovaných případů užití.

7.1 Jednotkové a *snapshot* testy

Bylo nutné testovat od nejnižších úrovní kvůli tomu, že se některé komponenty používají na několika místech aplikace. Příkladem je komponenta pro tlačítko `<CustomButton />`. Je efektivnější testovat tuto komponentu separovaně, nežli v rámci celku, kde by se mohla případná chyba hledat obtížněji. Za tímto účelem byly provedeny na všech „koncových" komponentách, které nemají žádné potomky, jednotkové (*unit*) testy. Testováno bylo jejich zobrazení v dokumentovém objektovém modelu a schopnost reagovat na přijaté `props`:

```
test('CustomButton rendered correctly', () => {
    const { getByTestId } = render(<CustomButton>Test</CustomButton>);
    expect(getByTestId("custom-btn-test")).toHaveTextContent("Test");
});
```

Společně s jednotkovými testy byly na stejné úrovni provedeny i *snapshot* testy a to především kvůli budoucí kompatibilitě a konzistenci. Mohlo by se stát, že by se při následných úpravách s příslušnou komponentou pracovalo nevhodně (a bez vědomí vývojáře). Tuto kompatibilitu ověřují právě *snapshot* testy:

```
test('CustomButton matches snapshot', () => {
    const tree = renderer.create(
        <CustomButton>TEST</CustomButton>
    ).toJSON();
    expect(tree).toMatchSnapshot();
})
```

7.2 Integrační testy

Pomocí integračních testů byly otestovány menší celky, jejichž funkčnost je pro implementovaný administrační portál zásadní. Zpravidla jde o testování takových komponent, které

jsou složené z několika dalších komponent, které se mohou vzájemně ovlivňovat. Prostřednictvím integračních, jednotkových a *snapshot* testů bylo celkově otestováno více než 40 vlastně vytvořených komponent. V případě integračních testů se jednalo např. o testování karty jednotky na stránce **Nástěnka** či tabulky pro prodlužování služeb.

7.3 Uživatelské testování

Uživatelské testování uživatelského rozhraní bylo prováděno společně se zástupci společnosti BIXION (cílovými uživateli) od raných stádií vývoje na pilotním nasazení celé aplikace, která byla hostovaná u komerčního poskytovatele. Na základě pravidelných online meetingů bylo uživatelské rozhraní průběžně vyvíjeno až do současné podoby. Finálně bylo uživatelské rozhraní otestováno na základě téměř 150 definovaných případů užití. Ty byly strukturovány do tabulky dle stránek, uživatelských rolí, událostí, požadované funkcionality a stavu (viz Obrázek 7.1). Soubor s jednotlivými případy užití se nachází na přiloženém paměťovém médiu.

Stránka	Role	Událost	Má se stát	Stav
Prostředí aplikace	Všechny	Kliknutí na tlačítko změny jazyka	Zobrazí se list dostupných jazyků	Passed
		Hover nad tlačítky v listu změny jazyka	Pozadí tlačítka se změní na světle šedou a kurzor se změní na pointer	Passed
		Kliknutí na tlačítko vybraného jazyka	Aktivní tlačítko jazyka je tmavě šedé a aplikace změní svůj jazyk na odpovídající jazyk	Passed
		Kliknutí na tlačítko odhlásit	Aplikace uživatele odhlásí	Passed
		Změna šířky okna pod 991 px a naopak	Pevné menu se změní na ovladatelný drawer s tlačítkem v hlavičce aplikace vpravo a naopak	Passed
		Kliknutí na tlačítko menu vpravo v hlavičce	Obrazovka ztmavne a vlevo vyjede správně zobrazený drawer	Passed
		Kliknutí mimo drawer, když je aktivní	Overlay obrazovky zmizí a drawer zajede mimo obrazovku	Passed
		Zobrazení hlavičky	Hlavička se drží na horní hraně viewportu a nad scrollujícím obsahem	To do
		Zobrazení pevného menu a draweru	Pevné menu i drawer se drží u levého okraje viewportu a mají na výšku 100 % výšky viewportu	To do
		Obsah menu zabírá více místa, než je výška viewportu	Menu i drawer jsou scrollable	To do
		Změny stránek s obsahem klikáním na tlačítka v menu	Menu i drawer se zobrazují správně na všech stránkách, obsah se mění pouze v pro něj vyhrazené části viewportu	Passed
		Navigace v aplikaci pomocí tlačítek prohlíče Zpět a Vpřed	Vše se zobrazuje jak má a historie stránek je ve správném pořadí	Passed
		Navigace v aplikaci pomocí tlačítek prohlíče Zpět a Vpřed nebo zadáním URL po kliknutí na tlačítko odhlásit	Uživatel je pokaždé přesměrován na stránku přihlášení	Passed
	Customer	Zadání url "/customers"	Vidí pouze prázdnou stránku	To do
		Zobrazení menu	Nevidí tlačítko "Customers" jinak všechno stejně	To do

Obrázek 7.1: Testování uživatelského rozhraní pomocí případů užití.

Kapitola 8

Závěr

Cílem bakalářské práce bylo navrhnout a vytvořit administrační portál pro časosběrný systém společnosti BIXION. Webová aplikace měla být vyvíjena takovým způsobem, aby v budoucnu umožňovala integraci dalších rozšíření. Dále měl být kladen důraz na modularitu, aby bylo možné některé části využít i v jiných projektech.

V rámci teoretické části byl na úvod představen pojem responzivní design. Poté bylo analyzováno několik frameworků a knihoven pro tvorbu responzivního uživatelského rozhraní webových aplikací, z nichž byla pro implementaci administračního portálu vybrána knihovna React. Ta byla podrobena detailní analýze. Následně byly nastudovány a popsány možnosti testování aplikací vyvíjených v Reactu.

Dále byl na základě popisu systému a požadavků na výslednou webovou aplikaci vytvořen návrh uživatelského rozhraní, podle kterého byl implementován portál pro administraci. Celkem bylo vytvořeno přibližně 80 vlastních komponent zajišťujících dostatečnou modularitu. Výsledná webová aplikace byla následně podrobena testům na několika úrovních. Postupně byly provedeny testy jednotkové, *snapshot*, integrační a také testy uživatelské.

Cíl této práce byl tedy úspešně splněn a navíc byla práce rozšířena o implementaci protokolu WebSocket a integraci digitální peněženky PayPal. Protokol WebSocket byl využit pro aktualizaci dat jednotek a zobrazování serverových notifikací v reálném čase. Digitální peněženka PayPal byla použita v rámci procesu prodlužování služeb, jež administrační portál poskytuje.

Na výstupy této bakalářské práce bude dále navazáno, neboť zástupci společnosti BIXION mají zájem na pokračování spolupráce i po jejím dokončení. Administrační portál tak bude dále vyvíjen a v dohledné době také produkčně nasazen. Možnými rozšířeními jsou integrace platební brány Stripe či využití protokolu WebSocket i pro aktualizaci dalších dat v aplikaci. Díky důkladnému návrhu a vhodně zvolené implementaci to bude s využitím již existujících komponent relativně jednoduché.

Literatura

[1] Ably Realtime. *WebSockets - A Conceptual Deep Dive.* [cit. 2021-28-04]. Dostupné z: https://ably.com/topic/websockets.

[2] Banks, A. a Porcello, E. *Learning React.* 2. vyd. Sebastopol: O'Reilly Media, Inc., 2020. ISBN 978-1-4920-5172-5.

[3] Bixion. *An intelligent intervalometer for capturing time-lapse images using special features.* [online]. 2019 [cit. 2021-01-05]. Dostupné z: https://www.bixicon.com/.

[4] Cazier, C. *Is Responsive Design A Ranking Factor?* [online]. Srpen 2015 [cit. 2021-21-01]. Dostupné z: https://searchengineland.com/responsive-design-ranking-factor-228464.

[5] Cloudaffle. *What Is ECMAScript And How Is It Different From JavaScript* [online]. Červen 2021 [cit. 2021-27-04]. Dostupné z: https://cloudaffle.com/post/what-is-ecmascript-how-different-from-javascript.

[6] Daityari, S. *Angular vs React vs Vue: Which Framework to Choose in 2021* [online]. Prosinec 2020 [cit. 2021-22-01]. Dostupné z: https://www.codeinwp.com/blog/angular-vs-vue-vs-react.

[7] Dodds, K. C. a přispěvatelé. *React Testing Library* [online]. 2021 [cit. 2021-01-05]. Dostupné z: https://testing-library.com/docs/react-testing-library/intro/.

[8] Existek. *Top Front-End Frameworks in 2020* [online]. Leden 2020 [cit. 2021-21-01]. Dostupné z: https://existek.com/blog/top-front-end-frameworks-2020/.

[9] Facebook Inc. *Jest* [online]. 2021 [cit. 2021-24-04]. Dostupné z: https://jestjs.io/.

[10] Facebook Inc. *React* [online]. 2021 [cit. 2021-23-04]. Dostupné z: https://reactjs.org/.

[11] Fette, I. a Melnikov, A. *The WebSocket Protocol* [Internet Requests for Comments]. RFC 6455. RFC Editor, prosinec 2011. 1-71 s. Dostupné z: https://tools.ietf.org/rfc/rfc6455.txt.

[12] Greif, S. a Benitte, R. *Technologies* [online]. [cit. 2021-30-04]. Dostupné z: https://2020.stateofjs.com/en-US/technologies/.

[13] Krotoffr, T. *Front-end frameworks popularity (React, Vue and Angular)* [online]. Leden 2021 [cit. 2021-21-01]. Dostupné z: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190.

[14] Majda, D. *Knihovny vs. frameworky* [online]. Listopad 2009 [cit. 2021-21-01]. Dostupné z: https://majda.cz/blog/265/.

[15] Michálek, M. *Marcotteho responzivní design* [online]. Srpen 2017 [cit. 2021-18-01]. Dostupné z: https://www.vzhurudolu.cz/prirucka/3-principy-rwd.

[16] Michálek, M. *Lazy loading v kontextu webového frontendu: co to je a proč to dělat?* [online]. Srpen 2019 [cit. 2021-19-04]. Dostupné z: https://www.vzhurudolu.cz/prirucka/lazy-loading.

[17] OpenJS Foundation. *What is npm?* [online]. Srpen 2011 [cit. 2021-22-04]. Dostupné z: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/.

[18] Sookocheff, K. *How Do Websockets Work?* [online]. Duben 2019 [cit. 2021-27-04]. Dostupné z: https://sookocheff.com/post/networking/how-do-websockets-work/.

[19] SPIR z. s. p. o. *Black Friday zvýšil návštěvnost internetových obchodů* [online]. Prosinec 2019 [cit. 2021-16-01]. Dostupné z: https://www.netmonitor.cz/black-friday-zvysil-navstevnost-internetovych-obchodu.

[20] Webpack. *Webpack* [online]. 2021 [cit. 2021-23-04]. Dostupné z: https://webpack.js.org/.

Příloha A

Obsah přiloženého paměťového média

- **/is-bixion/*** – Zdrojové soubory webové aplikace
- **/thesis/*** – Zdrojové soubory této bakalářské práce
- **/README.txt** – Detailní popis obsahu paměťovéh média a instrukce k instalaci
- **/ui-testing-sheet.xlsx** – Testování uživatelského rozhraní
- **/xdacer00.pdf** – Finální text bakalářské práce