Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

JavaScript na webovém serveru s využitím frameworku Node.js

Server-side JavaScript with Node.js runtime

Bakalářská práce

Vypracoval: Jan Svěrák

Vedoucí práce: PaedDr. Petr Pexa, Ph.D.

České Budějovice 2019

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan SVĚRÁK**
Osobní číslo:	**P15552**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obory:	**Anglický jazyk se zaměřením na vzdělávání**
	Informační technologie se zaměřením na vzdělávání
Název tématu:	**JavaScript na webovém serveru s využitím frameworku Node.js**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Cílem bakalářské práce je zpracování problematiky frameworku Node.js, jež umožnuje inovativní využití jazyka JavaScript na straně serveru. Tato technologie umožnuje používat stejný jazyk i stejné knihovny jak na straně webového serveru, tak na straně klienta (webového prohlížeče), kde je použití jazyka JavaScript tradiční a v podstatě nezastupitelné.

Teoretická část bude zaměřena na samotný Node.js. Budou popsány jeho vlastnosti, funkce, způsob práce, případy, kdy je vhodné jej použít, ale zároveň i ty, kdy je využití nevhodné. Zároveň bude popsán jeho balíčkovací systém NPM a práce s těmito balíčky. Využití jazyka JavaScript na straně severu bude především porovnáno s tradičně používanou serverovou skriptovací technologií PHP (ve spolupráci s databází MySQL), přičemž autor vyhodnotí, jaké výhody či nevýhody využití JavaScriptu na straně severu přináší.

Praktickým výstupem bude single-page webová aplikace určená pro organizaci práce v týmu. Bude disponovat funkcí chatu v reálném čase a systémem pro tvorbu a správu poznámek či seznamů s úkoly. Po prvotním načtení aplikace nebude potřeba stránku aktualizovat, všechna nová či změněná data budou načítána a přidávána na stránku dynamicky. Také změna aktuálně používané funkce nebude vyžadovat přechod na novou stránku resp. opakovanou interakci webového klienta a serveru.

Rozsah grafických prací: **CD ROM**

Rozsah pracovní zprávy: **40**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

1. **GOOGLE, INC. Chrome V8 [online]. [cit. 2017-04-22]. Dostupné z: https://developers.google.com/v8/**
2. **JOYENT, INC. Node.js [online]. [cit. 2017-04-22]. Dostupné z: https://nodejs.org/en/**
3. **NPM, INC. npm [online]. [cit. 2017-04-22]. Dostupné z: https://www.npmjs.com/**
4. **PHP GROUP. PHP: Hypertext Preprocessor [online]. [cit. 2017-04-22]. Dostupné z: https://secure.php.net/**
5. **THE MOZILLA FOUNDATION. JavaScript — MDN [online]. [cit. 2017-04-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript**

Vedoucí bakalářské práce: **PaedDr. Petr Pexa, Ph.D.**
Katedra informatiky

Datum zadání bakalářské práce: **24. dubna 2017**

Termín odevzdání bakalářské práce: **30. dubna 2018**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 24. dubna 2017

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

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

V Českých Budějovicích dne 15. dubna 2019.

Abstrakt/Anotace

Cílem bakalářské práce je zpracování problematiky frameworku Node.js, jež umožňuje inovativní využití jazyka JavaScript na straně serveru. Tato technologie umožňuje používat stejný jazyk i stejné knihovny jak na straně webového serveru, tak na straně klienta (webového prohlížeče), kde je použití jazyka JavaScript tradiční a v podstatě nezastupitelné. Teoretická část bude zaměřena na samotný Node.js. Budou popsány jeho vlastnosti, funkce, způsob práce, případy, kdy je vhodné jej použít, ale zároveň i ty, kdy je využití nevhodné. Zároveň bude popsán jeho balíčkovací systém NPM a práce s těmito balíčky. Využití jazyka JavaScript na straně serveru bude především porovnáno s tradičně používanou serverovou skriptovací technologií PHP (ve spolupráci s databází MySQL), přičemž autor vyhodnotí, jaké výhody či nevýhody využití JavaScriptu na straně severu přináší. Praktickým výstupem bude single-page webová aplikace určená pro organizaci práce v týmu. Bude disponovat funkcí chatu v reálném čase a systémem pro tvorbu a správu poznámek či seznamů s úkoly. Po prvotním načtení aplikace nebude potřeba stránku aktualizovat, všechna nová či změněná data budou načítána a přidávána na stránku dynamicky. Také změna aktuálně používané funkce nebude vyžadovat přechod na novou stránku, resp. opakovanou interakci webového klienta a serveru.

Klíčová slova

JavaScript, Node.js, NPM, PHP

Abstract

The aim of this bachelor thesis will be to develop and describe issues related to the Node.js framework, which enables the innovative use of JavaScript on the server-side. This technology allows us to use the same language and the same libraries on both the server-side and the client-side (web browser) where JavaScript is traditional and essentially irreplaceable. The theoretical part will focus on the Node.js. It will describe its properties, function, method of use, cases where it is appropriate to use it, but also where the use is inappropriate. Simultaneously, its NPM package system and how to work with these packages will be described. The use of JavaScript on the server-side will primarily be compared to the traditionally used PHP language (in collaboration with the MySQL database), and the author will evaluate the advantages and disadvantages of using JavaScript on the server-side. In the practical part, a single-page web application designed to organize team work will be created. It will feature real-time chat functionality and a system for creating and managing notes or task lists. After the app is first loaded, the page will not need to be refreshed, any new content or changed data will be loaded and added dynamically to the page. Also, changing the currently used function will not require a repeated interaction of web client and server.

Keywords

JavaScript, Node.js, NPM, PHP

Poděkování

Rád bych poděkoval vedoucímu práce, panu PaedDr. Petru Pexovi, Ph.D., za odbornou pomoc při zpracování mé bakalářské práce, jeho cenné rady a vstřícný přístup. Dále děkuji své rodině, bez které by mé studium na vysoké škole nebylo možné.

Obsah

1 Úvod

1.1 Východiska práce

Příchod frameworku Node.js přinesl možnost užití jazyka JavaScript na straně serveru a s tím i potenciální možnost využití stejného jazyka jak na straně serveru, tak na straně klienta. v takovém případě je teoreticky možné snížit počet technologií, nutných k osvojení za účelem vytvoření kompletní webové aplikace. Využití této technologie, může přinést řadu výhod, ale i přesto nemusí být využití Node.js vždy jednoznačnou volbou, vhodnou pro každý typ projektu.[1]

V současné době neexistuje na českém internetu aktuální průvodce použitím frameworku Node.js a jeho balíčkovacího systému NPM či srovnání jeho výhod a nevýhod s typicky využívanými technologiemi.

1.2 Cíle práce

Cílem bakalářské práce je zpracování problematiky frameworku Node.js, jež umožňuje používat stejný jazyk i stejné knihovny, jak na straně webového serveru, tak na straně klienta (webového prohlížeče), kde je použití jazyka JavaScript tradiční a v podstatě nezastupitelné.

Teoretická část bude zaměřena na samotný Node.js. Budou popsány jeho vlastnosti, funkce, způsob práce a případy, kdy je vhodné jej použít, ale zároveň i ty, kdy je využití nevhodné. Zároveň bude popsán jeho balíčkovací systém NPM a práce s těmito balíčky. Využití jazyka JavaScript na straně serveru bude především porovnáno s tradičně používanou serverovou skriptovací technologií PHP (ve spolupráci s databází MySQL), přičemž autor vyhodnotí, jaké výhody či nevýhody využití JavaScriptu na straně severu přináší.

Praktickým výstupem této práce bude single-page webová aplikace určená pro tvorbu a správu záznamů. Po prvotním načtení aplikace nebude potřeba stránku aktualizovat, všechny nová či změněná data budou načítána

a přidávána do stránky dynamicky. Změna obsahu tedy nebude vyžadovat přechod na novou stránku, resp. opakovanou interakci webového klienta a serveru.

Významnou částí aplikace, bude průvodce jejími funkcemi. Bude se jednat o seznam popisující funkce aplikace. Tento průvodce nebude zaměřen na funkčnost z pohledu uživatele aplikace, ale na fungování aplikace z pohledu vývojáře. Průvodce bude rozdělen do logických prvků pojednávajících o konkrétních částech aplikace jako je například mazání záznamu. Bude popisovat, jak je této funkcionality dosaženo a kde ve zdrojovém kódu můžeme tyto funkce najít. Zdrojový kód aplikace bude veřejně přístupný.

1.3 Metoda práce

V úvodu práce popíši co je framework Node.js, k čemu se používá a jak funguje. Dále se zaměřím na balíčkovací systém NPM a na práci s ním. Poté popíši jazyky JavaScript a PHP. Dále porovnám tyto jazyky mezi sebou, v kontextu použití na straně serveru. Získané poznatky prakticky demonstruji vytvořením webové aplikace používající výše zmíněné technologie a docílím tak jednotného použití skriptovacího jazyka JavaScript na straně serveru i klienta.

2 Node.js

Node.js je asynchronní, událostmi řízené JavaScriptové prostředí. Node.js je vyvíjen k účelu tvorby škálovatelných webových aplikací. Node.js je open-source a je možné ho používat zdarma pro osobní i komerční účely a běží na všech běžných platformách jako je Windows, Linux, Unix, Mac OSX atd.[1]

2.1 Proč používat Node.js

Node.js využívá asynchronního programování. Typický úkon webového serveru může být otevření souboru na server a vrácení obsahu téhož souboru klientovi.[2]

PHP vykoná tento úkon následovně:

1. Odešle úkol do souborového systému počítače.
2. Čeká, než souborový systém otevře a přečte soubor.
3. Vrátí obsah souboru klientovi.
4. Je připraven vykonat další úkol.[2]

Node.js vykoná úkol následovně:

1. Odešle úkol do souborového systému počítače.
2. Je připraven vykonat další úkol.
3. Jakmile souborový systém otevře a přečte soubor, vrátí obsah klientovi.[2]

Node.js eliminuje fázi čekání a okamžitě pokračuje k vykonání dalšího úkolu. Hlubšímu porovnání využití Node.js oproti PHP na straně serveru je věnována kapitola číslo 3.

Node.js používá jedno vlákno a využívá neblokujícího, asynchronního programovaní, což je vysoce efektivní z pohledu operační paměti.[2]

2.2 Co dokáže Node.js vykonávat

S pomocí Node.js můžeme generovat dynamický obsah stránky. Vytvářet, číst, přepisovat, odstraňovat a zavírat soubory na webovém serveru. Také můžeme sbírat data z formulářů, či přidávat, upravovat a mazat data v databázi.[2]

Node.js je postaven na JavaScriptovém enginu „V8“.

2.2.1 V8 engine

V8 je vysoce výkonný open-source JavaScriptový engine vyvíjený společností Google.

Samotný V8 engine je napsán v jazyce C++ a kromě Node.js jej využívá například webový prohlížeč Google Chrome.[3]

2.3 Jaké firmy v praxi využívají Node.js

Za indikaci použitelnosti a kvality tohoto frameworku můžeme považovat fakt, že jej v praxi na svých produktech využívá množství známých, technologicky i jinak zaměřených firem, které otevřeně mluví o svých zkušenostech s Node.js.[4]

Následuje několik příkladů firem využívající Node.js. Některé svůj produkt od začátku vyvíjí s pomocí Node.js a jiné svůj stávající produkt na Node.js migrují. Za zajímavý můžeme považovat fakt, že firmy mají různé důvody, k využívání Node.js.[4]

2.3.1 Netflix

Netflix je přední poskytovatel služby pro streamování video-obsahu. Na své platformě silně využívají A/B testování v rámci optimalizace uživatelského zážitku pro 93 miliónů odběratelů jejich služby, což znamená velké množství jedinečných balíčků dat. Rozhodli se tedy využít rychlosti a asynchronnosti Node.js. Jeden z hlavních přínosů bylo zkrácení doby prvotního spuštění jejich aplikace a to o 70 procent.[4]

2.3.2 Treollo

Trello je aplikace určená k managementu projektů pro vícečlenné týmy lidí. Aplikace umožňuje vytváření úkolů, či seznamu úkolů a uživatelé očekávají okamžitou aktualizaci stavu jednotlivých úkolů. Aplikace si tedy vyžaduje držení několika aktivních připojení zároveň. Právě z těchto důvodů je serverová strana projektu Trello vyvinutá s pomocí Node.js.[4]

2.3.3 Paypal

Paypal je celosvětový systém pro online platby, jehož back-end byl původně napsán v jazyku Java. Dnes už se ale jejich vývoj přesunul na Node.js a tedy JavaScript. Jeden z důvodů je ten, že vývojáři byli rozděleni do dvou skupin. Jedna z nich programovala aplikační vrstvu a druhá psala kód určený pro prohlížeč. Takový souběžný vývoj dvou skupin používající rozdílné technologie, nebyl ideální. Přechod na Node.js umožnil používání stejného jazyka jak na straně klienta, tak na straně serveru. Umožnil sjednocení obou týmů vývojářů, což napomohlo k vzájemnému porozumění a přineslo zvýšení efektivity vývoje, a tedy rychlejší reakce na potřeby zákazníků.[4]

2.4 První spuštění jednoduchého webserveru

K založení prvního webserveru po nainstalování Node.js stačí jeden soubor s kódem:[5]

```
1  const http = require('http');
2  const hostname = '127.0.0.1';
3  const port = 3000;
4
5  const server = http.createServer((req, res) => {
6    res.statusCode = 200;
7    res.setHeader('Content-Type', 'text/plain');
8    res.end('Hello World\n');
9  });
10
11 server.listen(port, hostname, () => {
12   console.log(`Server running at http://${hostname}:${
       port}/`);
13 });
```

Ukázka kódu 1: Příklad jednoduchého webserveru, Hello world projekt

Tento jednoduchý příklad využívá pouze balíček nesoucí jméno http. Tento balíček nemusíme sami instalovat, je již součástí Node.js. Po spuštění serveru v příkazové řádce uvidíme lokální adresu, na které uvidíme výsledek provedeného kódu. v našem případě, se bude jednat o výpis textu „Hello World“.[5]

3 JavaScript vs PHP při využití na serveru

Odpověď na otázku, jaký je rozdíl mezi JavaScriptem a PHP, byla před deseti lety relativně jednoduchá. JavaScript je skriptovací jazyk určený pro stranu klienta a PHP je skriptovací jazyk určený pro stranu server. Společně tvoří dynamické interaktivní webové stránky. To se změnilo s příchodem Node.js, jež umožnil, využití JavaScriptu i na straně serveru. Proto je vhodné tyto jazyky blíže porovnat. v našem kontextu tedy mluvíme o rozdílech, mezi použitím PHP a Node.js na straně serveru.[6, 7]

3.1 Společné znaky

Oba dva tyto jazyky jsou vhodnou volbou při tvorbě webových stránek a aplikací. Je možné jejich kód spouštět na běžně dostupných hostingových řešení. v obou případech se jedná o jazyky interpretované, což znamená že napsaný kód není potřeba překládat a server, případně prohlížeč, jej rovnou provádí.[7]

Tyto jazyky jsou všudypřítomné. Společně tyto jazyka pohání většinu webových stránek a aplikací. To znamená, že kolem obou jazyků je rozsáhlá komunita a pro oba jazyky existuje mnoho knihoven a frameworků. Výhodou využívání populárního jazyka může být ta skutečnost, že v případě naražení na problém, je snadné najít člověka, jež narazil na stejný problém a dokáže poradit, jak tento problém vyřešit.[7, 8]

Node.js i PHP mají vlastní balíčkovací systém. v případě PHP jde o „Composer“, Node.js má vlastní systém NPM. Oba tyto systémy slouží k centralizaci balíčků určených k usnadnění práce, využití již existujícího otestovaného kódu atd. o balíčcích a systému NPM blíže pojednává kapitola číslo 4.[8]

3.2 Kdy využít Node.js na straně serveru

Node.js může být dobrou volbou pro aplikace, které vyžadují zpracování velkého množství malých celků dat, kde je podmínkou nízká latence. Takové sys-

témy jsou tzv. „real-time aplikace“. Příkladem mohou být různé kolaborační aplikace, které umožňují více uživatelům upravovat stejné dokumenty ve stejnou dobu (např. Google Docs). Dalším příkladem může být chatovací aplikace, či software pro videokonference. Node.js může být díky své povaze použit i při tvorbě aplikací, u kterých je zapotřebí velice rychlá odezva, jako například systém online aukcí, či dokonce online multiplayerové hry.[9]

Node.js je vhodný pro výše zmíněné real-time aplikace díky tomu, že dokáže zpracovávat úkony z více klientů v tu samou chvíli a synchronizace dat mezi klientem a serverem je velice rychlá.[9]

3.3 Kdy je využití Node.js nevhodné

Jak již bylo zmíněno, Node.js je založen na událostmi řízeném modelu a využívá jedno procesorové vlákno. z toho důvodu není vhodné využít Node.js v případě, že aplikace bude náročná právě na procesorové operace. Pokud bude aplikace provádět náročné výpočty, zablokuje se tím možnost vykonávat ostatní příchozí úkoly, čímž bude aplikace ochuzena o hlavní výhodu Node.js. Avšak na takovéto náročné operace u běžných webových stránek a aplikací narazíme jen zřídka.[9]

Oproti tomu jeho přímý konkurent jazyk PHP, využívá při práci více vláken Ve výchozím stavu nepracuje asynchronně, a tedy vykonání stejné sady operací prováděných na jednom vlákně, může být s využitím PHP pomalejší než s využitím Node.js, avšak PHP přináší výhodu využívání více vláken a tedy pokud je jedno vlákno zaneprázdněné časově náročnější operací, je možné aby další operace byla vykonávána na vlákně jiném. Skutečností ale zůstává, že více vláken také vyžaduje větší výpočetní výkon.[7, 9]

3.4 Výhody využití Node.js oproti PHP

Pro fungování PHP na serveru je potřeba přítomnost softwarového webového serveru, jako příklad může posloužit, typický využívaný, Apache webový server. Oproti tomu Node.js nic takového nevyžaduje. Pro fungování na serveru

využívá vlastního běhového prostředí, které je přímo k tomuto účelu vytvořené a je nedílnou součástí Node.js jako takového.[8]

Nespornou výhodou je již zmiňované použití stejného jazyka v celé aplikaci, tedy jak ve frontendové tak i backendové části. Toto je výhodné zejména při tvorbě dynamických sigle-page aplikací, či webových stránek s renderováním na straně serveru.[8]

Další výhodou využití Node.js je jeho rychlost. PHP je oproti Node.js primárně synchronní jazyk. Možnost asynchronního provádění kódu a přístupu k datům v reálném čase může zaznamená svižnější fungování aplikace oproti takové, která bude vytvořena za pomocí jazyka PHP.[8]

3.5 NoSQL vs MySQL

Při tvorbě webových aplikací s využitím jazyka PHP je běžné použití MySQL databáze. Oproti tomu u Node.js aplikací jsou, vzhledem k povaze těchto projektů, běžně používané takzvané „NoSQL“ databáze. v praktické části této práce využívaná „Firebase Realtime Database“ je příkladem takové databáze.[10]

3.5.1 Datový model

Zásadním rozdílem, mezi NoSQL a MySQL databází, je využívaný datový model. MySQL je relační databáze, kde jsou základním konstruktorem tabulky, které obsahují jednotlivé řádky (záznamy). Mezi daty se vytvářejí pomocí klíčů relace (vztahy) a jednotlivé tabulky je poté možné, na základě definované relace, spojovat do jednoho logického celku. Finální spojení více tabulek musí být uskutečněno na jednom uzlu a při přenosu velkého množství dat může tento proces být relativně pomalý.[11]

NoSQL je ne-relační databáze, kde jsou data uspořádána do tzv. „dokumentů“. Použitá Firebase Realtime Database využívá k úschově dat stromovou strukturu ve formátu JSON. Jelikož JSON je zápis dat po vzoru JavaScriptového

objektu (JSON = JavaScript Object Notation), je práce s touto databází ve spojení s Node.js a tedy JavaScriptem velmi snadná.[11]

Ne-relační databázový model byl původně vytvořen jako opak relačních databází. Jeho cílem je dostupnost dat s co nejkratší odezvou. NoSQL používá přístup k transakcím, který není tak striktní jako ten, který využívá MySQL. To má za výsledek rychlejší odezvu databáze za cenu dočasné inkonzistence v uložených datech.[11]

V případě MySQL se jedná o přístup ACID. Transakce tedy musí proběhnout celá, nebo vůbec. Transakce převede databázi z jednoho konzistentního stavu do druhého a pokud je změna úspěšná, zůstane na trvalo uložena v databázi. u NoSQL se jedná o přístup BASE. Tento přístup není tolik striktní jako ACID, jelikož nevyžaduje konzistenci pro celou dobu transakce, vyžaduje pouze eventuální konzistenci v konečném stavu. Za inkonzistenci v tomto případě považujeme dočasnou přítomnost neaktuálních dat v některých částek databáze. Díky tomu ovšem dokáže NoSQL databáze velmi rychle přistupovat k datům, pouze za cenu případně inkonzistence mezi uzly. Takovou databázi je možné velmi snadno horizontálně škálovat a lineárně tak zvyšovat výkon databáze vzhledem k rychlému nárůstu dat. NoSQL databáze jsou tedy vhodné i při práci s obrovským množstvím rychle se měnících dat.[11]

3.5.2 Škálovatelnost

MySQL databáze nabízí především vertikální škálování a tedy škálování kvalitativní. Tím je myšlena kvalitativní změna stávajících prvků systému s cílem zvýšit výkon. v praxi to znamená upgrade hardwaru serveru, či zvýšení propustnosti síťového připojení atd.[11]

Oproti tomu NoSQL databáze nabízí horizontální škálovatelnost, a tedy škálovatelnost kvantitativní. To znamená, zvýšení výkonu za pomocí dalšího hardware, např. ve formě dalšího samostatného serveru, pokud si to aplikace zrovna vyžaduje.[11]

V praxi to může znamenat zaplacení většího počtu výpočetních jednotek serverů, jež si pro běh aplikace pronajímáme. Na této vlastnosti je důležité to, že tohoto zvýšení výkonu můžeme využívat i pouze dočasně.[12, 13]

Typickým příkladem může být očekávaný nárůst přístupu uživatelů k aplikace. Představme si situaci, kdy firma provozuje stránku, na které se mohou uživatelé registrovat do soutěže. Tato stránka nemá běžně přístup stovek lidí zároveň a vůči tomu je i nastaven výpočetní výkon, jež tuto stránku pohání. Pokud v televizi proběhne reklama, jež upozorní diváky na existenci této soutěže a pobídne je k registraci na dané stránce, je možné že výkon bude pro náhlý vzrůst aktivity uživatelů nedostačující. Právě pro tuto situaci je vhodné zvýšit výpočetní výkon dané aplikace, což s sebou přináší dočasné zvýšení nákladů na provoz, a po opadnutí tohoto návalu opět přejít na původní konfiguraci.[13]

Řada poskytovatelů služeb tohoto typu, jako jsou například AWS, Azure a Google Cloud Platform nabízí možnost tohoto dočasného zvýšení poskytovaného výkonu, a to i automaticky v případě zvýšené aktivity.[12]

4 NPM

NPM (Node package manager) je, jak již z název napovídá, balíčkovací systém pro Node.js.

NPM se skládá ze tří částí:

1. Webová stránka
2. Command Line Interface (CLI)
3. Registr [14]

4.1 Webová stránka

Webová stránka `www.npmjs.com` slouží jako prostředek k objevování a procházení balíčků. Je možné si zde založit uživatelský účet, který mimo jiné přináší možnost publikovat vlastní open-source kód do NPM registru a tím se podílet na dalším vývoji balíčků pro Node.js. Kromě účtu pro jednotlivce je možné také vytvořit tzv. „Orgs“ účet, a tedy účet organizace, který umožňuje organizovat balíčky a týmy do skupin s určitými právy vztahující se mi k balíčkům vytvořených organizací.[14]

4.2 Command Line Interface

CLI - prostředí pro příkazovou řádku umožňuje pracovat s balíčky přímo v příkazové řádce. Jedná se také o nejběžnější způsob, jak vývojáři s NPM pracují.[14]

4.3 Registr

Registr NPM je rozsáhlá, veřejná databáze JavaScriptového softwaru. i přesto, že registr NPM je primárně veřejná databáze, neznamená to, že všechny balíčky zde nahrané, musí nutně být veřejně přístupné. Tvůrce balíčků se může rozhodnout, zda budou veřejně přístupné, nebo zda bude mít přístup jen konkrétní uživatel, či organizace.[14]

4.4 Balíček

Balíček je ve své podstatě malý kus kódu, jež řeší jeden konkrétní problém, ale řeší ho velmi dobře. Balíček umístěný v databázi NPM by nikdy neměl řešit více problémů najednou. Důvod k tomu je takový, že pokud jeden balíček vyřeší konkrétní problém, jiný balíček vyřeší jiný konkrétní problém a tak dále. Tím vzniká modulární systém, jež napomáhá znovupoužitelnosti kódu.[14]

Díky tomu, že balíček vždy řeší jeden konkrétní problém, jej mohou psát profesionálové pro velice specifickou oblast. To znamená, že výsledek může být mnohem kvalitnější, právě proto, že pro konkrétní problém, použijeme kód napsaný odborníkem na danou problematiku, a tedy kód kvalitnější. Daný odborník poté může využít ve své aplikaci balíček jiného odborníka, protože balíček řeší problém v oblasti mimo jeho expertízu.[14]

I v případě, že se organizace rozhodne, že všechny balíčky jež vytvoří nebudou veřejně dostupné, a tedy je budou smět používat pouze členové organizace, může být tento přístup k tvorbě software přínosný, díky výše zmíněné znovupoužitelnosti kódu.[14]

Pokud firma tvoří software tímto přístupem a v minulosti vytvořila několik aplikací, jež se skládají s různých balíčků, je pravděpodobné, že v budoucnosti budou tvořit aplikaci, kde budou některé části stejné jako v minulých projektech. Díky systému balíčků si mohou jednoduše vybrat ty, které pro tento konkrétní projekt budou potřebovat.[14]

Další výhodou balíčku je, že pracují s tzv. závislostmi. To znamená, že v projektu můžeme nastavit závislosti na určitém balíčku z databáze NPM a pokud ho jeho tvůrce aktualizuje, budu mít i já ve svém projektu, jeho aktuální verzi.[14]

Balíčky jsou ve své podstatě trojího druhu, dle způsobu jejich použití.

1. Balíčky pro využití na serveru.

2. Balíčky pro využití v příkazové řádce.

3. Balíčky pro použití v prohlížeči.[14]

4.4.1 Balíčky pro využití na serveru

Jedná se o balíčky určené pro využití na webovém serveru, na kterém používáme Node.js framework. Typickým příkladem takového balíčku je „request“, což je velice oblíbený, jednoduchý a hojně využívaný balíček vytvořený za účelem umožnit, co nejjednodušší realizaci HTTP volání, který zároveň podporuje zabezpečenou variantu HTTPS.[14, 15]

4.4.2 Balíčky pro využití v příkazové řádce

Takové balíčky mohou např. rozšiřovat funkcionalitu příkazové řádky. Příkladem může být balíček „jshint“, což je nástroj pro analýzu statického JavaScriptového kódu. Jeho cílem je hledat chyby v kódu, což může potenciálně ušetřit hodiny strávené laděním. Dokáže tedy procházet napsaný kód a pomocí příkazové řádky hlásit typické chyby jako nedodržení syntaxe, či překlepy.[14, 16]

4.4.3 Balíčky pro použití v prohlížeči

Jde o balíčky, jež pomáhají na straně klienta, a tedy v prohlížeči. Typickým příkladem je balíček „js-cookie“. Jedná se o jednoduché JavaScriptivé API, které usnadňuje práci s cookies a funguje ve všech relevantních prohlížečích.[14, 17]

4.4.4 Co je to balíček z technického pohledu

Balíček NPM je ve své podstatě adresář obsahující samotný program, jež je popsaný pomocí speciálního souboru „package.json“. o package.json souboru hovoří kapitola číslo 5.[18]

4.5 Bezpečnostní problémy NPM balíčků

Používání open-source balíčků sebou může nést bezpečnostní rizika. Při využívání NPM balíčků, je typické, že v projektu je přítomné velké množství kódu, jehož autorem je třetí strana. Na tento kód se uživatelé balíčků často ani nepodívají. To může přinést bezpečnostní rizika, proto je vhodné se při výběru a používání balíčků třetích stran řídit několika pravidly.[19, 20]

4.5.1 Kontrola popularity balíčku

Pokud hledáme balíček, jež má řešit některé z našich problémů je možné, že narazíme na více balíčků se stejným účelem. v takovém případě může být vhodné rozhodnout se pro ten, jenž je více populární. Pokud je jeden z balíčků využíván znatelně větším množstvím dalších vývojářů, je pravděpodobné, že i vám bude vyhovovat. Vybírat konkrétní balíček na základě popularity s sebou může přinést i další benefity viz níže.[20]

4.5.2 Kontrola aktivity vývojářů balíčku

I když popularita u uživatelů může být dobrým ukazatelem kvality balíčku, aktivita jeho tvůrců může být ještě důležitější. Před instalací balíčku je vhodné podívat se do jeho historie. Pokud první verze balíčku nebyla zveřejněna teprve nedávno a balíček má přesto jen velmi malý počet dalších verzí, pravděpodobně tvůrce balíček aktivně nepodporuje. Velká část bezpečnostních problémů je spjata s chybou v kódu balíčku a v případě, že je taková bezpečnostní mezera objevena, je žádoucí, aby jí tvůrce co nejdříve opravil.[20]

Pokud tvůrce není příliš aktivní, může se stát, že taková chyba zůstane dlouhou dobu neopravena.[20]

4.5.3 Kontrola známých problémů s balíčkem

Portál npmjs.com přímo na stránkách jednotlivých balíčků umožňuje uživatelům nahlásit jejich bezpečnostní problémy. Stejně tak Github, kde je umístěn

zdrojový kód většiny NPM balíčků, umožňuje uživatelům veřejně nahlašovat problémy v daném repozitáři.[20]

Pokud je tedy delší dobu veřejně známá bezpečnostní chyba v balíčku, a tvůrce tohoto balíčku nijak nereaguje, je těžké takovýto balíček doporučit.[20]

Pokud je balíček populární, zvedá se tím i šance, že někdo takovou chybu objeví a nahlásí. Avšak zde již můžeme vidět, že není vhodné se slepě řídit popularitou balíčku, ale prověřovat i jeho další aspekty. Může nastat situace, kdy si balíček v minulosti nainstalovalo velké množství uživatelů a jehož autor byl v minulosti aktivní. Pokud ale vidíme, že u balíčku je veřejně známý bezpečnostní problém, autor tuto situaci nijak nevyřešil a poslední verze balíčku je již staršího data, je pravděpodobně rozumné poohlédnout se po balíčku jiném.[20]

4.5.4 Kontrola aktuálnosti verze balíčku

Pokud tvůrce balíček aktivně vyvíjí, udržuje jeho kód a řeší jeho případné bezpečnostní problémy, neznamená to, že je náš konkrétní způsob využití tohoto balíčku bezpečný.[20]

V kapitole číslo 5 jsou zmíněny různé zápisy vyžadované verze balíčku. Jak je zmíněno, poslední ze tří číslic znační opravy chyb, jež nesmí ovlivnit zpětnou kompatibilitu balíčku. Pokud tedy udáme jako podmínku konkrétní verzi balíčku, např. 1.0.2 a v této verzi je objevena bezpečnostní chyba, i rychlá reakce tvůrce ve formě opravy a vydání verze 1.0.3 nevyřeší chybu v našem projektu. z toho důvodu je vhodné uvádět číslo požadované verze alespoň pomocí zápisu 1.0.x.[20]

4.5.5 Odstranění nepoužívaných balíčků

Jak již bylo zmíněno výše, dnes bezpečný a udržovaný balíček, nemusí být stejně aktuální a bezpečný v blízké budoucnosti. Je tedy vhodné balíčky, jež využíváme průběžně kontrolovat na základě zmiňovaných pravidel. s větším

množstvím balíčků se zvyšuje náročnost tohoto procesu. s větším počtem balíčků se zároveň zvyšuje pravděpodobnost přítomnosti bezpečnostní chyby.[20]

Z toho důvodu je vhodné ohlédnout se, zda všechny nainstalované balíčky stále využíváme a nevystavujeme se tak potencionální bezpečnostní hrozbě zbytečně.[20]

Není od věci zmínit, že bezpečnostní problémy nemusí přinést jen přímé závislosti našeho projektu. Je velice pravděpodobné, že balíčky, jež v projektu využíváme mají samy několik závislostí na jiné balíčky. Pokud se tedy objeví bezpečnostní chyba v těchto balíčcích, mohou potencionálně zasáhnout i náš projekt.[20]

4.5.6 Github security alerts

Pokud využíváme při tvorbě našeho projektu Github, jednou z výhod je vlastnost týkající se hlášení známých bezpečnostních problémů koncovým uživatelům balíčků. Github sleduje nahlášené bezpečnostní problémy závislostí projektů a vlastníkům repozitářů poskytuje emailové upozornění na tyto chyby. Vlastník repozitáře je tak obeznámen s přítomností potencionálního bezpečnostního problému a je přímo odkázán na konkrétní balíček. Pak je již na každém vývojáři, jak s onou informací vynaloží.[21]

4.6 Instalace NPM balíčků

Typickým postupem instalace NPM balíčků je využití CLI zmiňovaného výše. NPM CLI nám umožní v příkazové řádce využívat NPM příkazy, nejjednodušším způsobem instalace NPM balíčku je příkaz:[22]

```
npm install
```

Tento příkaz zavoláme v adresáři, kde se nachází soubor package.json. v témže adresáři bude vytvořena složka „node_modules“, do níž jsou posléze nainstalovány všechny balíčky, jež jsou v souboru package.json uvedeny jako závislosti.

Nainstalované balíčky jsou poté dostupné pouze v kontextu tohoto adresáře a jeho potomků.[22]

K příkazu npm install můžeme přidávat množství argumentů, či „vlajek“, jež ovlivní přesné chování tohoto příkazu.[22]

Příkladem příkazu s takovým argumentem může být dvojice příkazů:[22]

```
npm install -g
npm install --global
```

Pokud zavoláme libovolný z těchto dvou příkazů, závislosti z package.json v kontextu adresáře, ze kterého je příkaz volán, jsou nainstalované globálně. To umožňuje využívání funkcí balíčků ve více projektech na našem zařízení, bez nutnosti instalace v každém adresáři, kde chceme balíčky použít.[22]

Pokud chceme při instalaci vynechat balíčky vypsané v seznamu devDependencies, použije příkaz:[22]

```
npm install --production
```

Po zavolání tohoto příkazu jsou při instalaci vynechány balíčky vypsané v seznamu devDependencies a tedy balíčky určené pro vývoj projektu. Instalací pouze těch balíčků, jež jsou potřebné pro funkci projektu u koncového uživatele můžeme ušetřit úložný prostor a zkrátit dobu instalace.[22]

Při instalaci balíčku, který dosud nebyl součástí projektu a tedy není z tohoto, či jiného důvodu uveden jako závislost v souboru package.json, uvádíme jako argument příkazu jeho jméno.[22]

Pro nainstalování balíčku pro práci se službou Firebase od společnosti Google použijeme příkaz:[22]

```
npm install firebase
```

Pokud při použití tohoto příkazu není uveden další argument, tento balíček je automaticky přidán v souboru package.json jako závislost do seznamu dependencies, tedy jako uživatelská závislost projektu.[22]

Pokud chceme balíček uložit do seznamu devDependencies a tedy jako vývojářskou závislost, využijeme příkaz:[22]

```
npm install firebase --save-dev
```

V případě, že chceme balíček zapsat do seznamu volitelných závislostí, zavoláme příkaz:[22]

```
npm install firebase --save-optional
```

Pokud chceme balíček nainstalovat, ale nepřejeme si, aby byl uložen jako závislost v package.json, použijeme použijeme příkaz:[22]

```
npm install firebase --no-save
```

5 Package.json

Package.json je nedílnou součástí práce s Node.js a NPM. Tento soubor je jakýmsi manifestem Node.js aplikace. Package.json obsahuje základní informace o projektu, jeho konfiguraci a případných závislostech na jiných NPM balíčcích. Jedná se tedy o jakousi kolekci meta-informací o projektu napsanou striktně ve formátu JSON.[23, 24]

Za zajímavou můžeme považovat tu skutečnost, že žádná z dále uvedených, možných meta-informací není povinná. Výjimkou jsou pouze jméno a verze projektu, a to pouze v případě, že konkrétní projekt, či balíček má být veřejně publikovatelný.[24]

```
{
  "name": "sverak-notes-app",
  "version": "1.0.0",
  "author": "Jan Sverak <info@jansverak.cz> (http://
     jansverak.cz)",
  "description": "Node.js notes demo app",
  "keywords": [
      "Notes",
      "Firebase",
      "Vue"
    ],
  "homepage": "https://sverak-notes-app.firebaseapp.com
     ",
  "devDependencies": {
      "webpack": "^1.12.2",
      "browser-sync": "^2.24.7",
    },
  "dependencies": {
      "firebase": "^5.7.1",
      "vue": "^1.0.18",
    }
}
```

Ukázka kódu 2: Příklad souboru package.json

5.1 Základní informace o projektu

První sadou meta-informací jsou položky vztahující se k základnímu popisu projektu.

5.1.1 Name

Definuje jméno projektu. Délka jména nesmí přesahovat 214 znaků, což není výzvou k vymýšlení jmen této délky. Jméno by mělo být krátké a zároveň dostatečně deskriptivní.[24, 25]

Jméno nesmí začínat podtržítkem či tečkou. Jméno projektu je v praxi používáno jako součást URL adresy, argument v příkazové řádce a jako jméno složky, z toho důvodu je nutné při pojmenovávání projektu používat jen znaky, jež jsou vhodné pro URL adresy. Doporučuje se tedy vyvarovat se speciálním znakům jako jsou mezery a samozřejmě znakům specifických pro český jazyk.[25]

Dokumentace NPM doporučuje řídit se při výběru jména a několika pravidly, pokud je naším projektem novým balíček, jež plánujeme publikovat na portálu npmjs.com.[25]

Nedoporučují se taková jména, která se shodují se jmény balíčků z jádra Node.js. Dále se nedoporučuje uvádět jako součást jména výrazy jako „js“ a „node“. Výraz „js“ je považován za redundantní, jelikož použití JavaScriptu je při psaní package.json zjevná (JSON = JavaScript Object Notation). Pro specifikaci, že je balíček určen pro Node.js je připravena položka „engine“. Zároveň se silně doporučuje zkontrolovat v registru NPM, zda již neexistuje balíček s totožným jménem, jaké máme v plánu použít.[25]

5.1.2 Version

Udává verzi projektu. Pokud se chystáme náš projekt publikovat jako balíček, jsou jeho jméno a verze nejdůležitější, a zároveň povinné, informace. Verze pro-

jektu je vyjádřena číselnou hodnotu a měla by se změnit s každou aktualizací projektu.[25]

5.1.3 Author

Informace o autorovi projektu. Jedná se o textový řetězec, který typicky obsahuje jméno, emailovou adresu a odkaz na webové stránky autora.[25]

Autor se zapisuje jako objekt:

```
"author": {
  "name": "Jan Sverak",
  "email": "info@jansverak.cz",
  "url": "http://www.jansverak.cz"
}
```

Ukázka kódu 3: Package.json - autor projektu

Případně je možné použít zkrácený zápis:

```
"author": "Jan Sverak <info@jansverak.cz> (http://
   jansverak.cz)",
```

Ukázka kódu 4: Package.json - autor projektu, zkrácený zápis

V tom případě je emailová adresa obalena ostrými závorkami a URL adresa závorkami kulatými. Položky emailové adresy a URL webové stránky nejsou povinné.[25]

Toto pole je určené pouze pro zápis jedné osoby.[25]

5.1.4 Contributors

Přispěvatelé projektu. Jedná se o pole záznamů, jehož jednotlivé položky jsou složené ze stejných částí jako obsah informace „author“. Tuto informaci udáváme v případě, že projekt má více tvůrců, nebo v případě, že má jednoho hlavního tvůrce, tedy autora, a několik dalších přispěvatelů.[25]

5.1.5 Description

Udává krátký popis projektu. Mělo by se jednat o velmi stručný popis daného projektu ve formě řetězce. v případě publikování na npmjs.com je využíván při vyhledávání balíčků.[25]

5.1.6 Keywords

Definice klíčových slov. Jedná se o pole řetězců s klíčovými slovy vztahujících se k danému projektu. Užitečné hlavně v případě publikování balíčku na npmjs.com, jelikož je součástí vyhledávání balíčků.[25]

5.1.7 Homepage

Odkaz na domovskou stránku projektu. v případě že projekt nemá vlastní webové stránky, je typicky uvedena veřejná adresa repozitáře např. na portálu github.com. v tom případě je uživatel při přechodu na tuto adresu uvítán souborem readme.md, kde se tradičně může dozvědět více podrobností o daném projektu.[25]

5.1.8 Licence

Udává licence projektu, či balíčku. Pro balíčky šířené na npmjs.com jsou typické licence „MIT“, „ISC“ a „GPL-3.0“.[25]

5.1.9 Private

Tato položka nabývá pouze hodnot true nebo false. Zde udáváme, zda je projekt privátní, či nikoliv. Hodnotou false lze zamezit nechtěnému, či předčasnému publikování projektu na npmjs.com.[25]

5.2 Technické informace o projektu

Další meta-informace jsou položky vztahující s k technické realizaci projektu.

5.2.1 Main

Vstupní bod aplikace. Zde zadáváme cestu k vstupnímu bodu projektu či balíčku. Typicky se tedy jedná o JavaScriptový soubor, kde najdeme exporty použitých modulů. Jedná se právě o soubor, který je výchozím bodem při spuštění aplikace. u balíčků většinou není potřeba mít JavaScriptový kód rozdělen do více souborů a „mail“ tedy směřuje na samotný main.js. u komplexnějších aplikací bývá zvykem přítomnost souboru app.js, který je spíš jakýmsi rozcestníkem pro jednotlivé části aplikace, reprezentované jednotlivými soubory.[25]

5.2.2 Závislosti

Další sadou informací jsou závislosti. Závislostí je míněno, jaké balíčky jsou využívané tímto projektem a tedy, bez jakých balíčků nemůže správně fungovat. Závislosti mohou být dvojího typu, „dependencies“ a „devDependencies“.[25]

První ze dvou variant (dependencies), se považuje za uživatelské závislosti. Tím jsou myšlené ty balíčky, jejichž funkce budou využívány koncovými uživateli aplikace, či balíčku. Jedná se o balíčky, které jsou potřebné pro samotnou funkčnost projektu.[25]

Druhá varianta (devDependencies) je určena pro výpis balíčků, které nejsou pro koncové uživatele přínosné, či nezbytné. Jedná se tedy o vývojářské balíčky, určené k usnadnění práce lidem pracujícím se zdrojovým kódem daného projektu.[25]

Toto rozdělení je potřebné z důvodu zamezení uživateli zbytečně stahovat balíčky, které při běhu aplikace nebudu používat.[25]

Je možné, že celý projekt je určený jako pomůcka, či nástroj pro další vývojáře. v tom případě je reálné, že bude přítomen pouze výpis „devDependencies“. Obdobně je možná i přítomnost samotného „dependencies“. v případě že pro-

jekt nevyužívá žádných externích balíčků, není nutná přítomnost ani jednoho z těchto polí.[25]

Zápis jednotlivých závislostí v obou případech je následující:

```
"dependencies": {
  "foo": "1.0.0 - 2.9999.9999",
  "bar": ">=1.0.2 <2.1.2",
  "baz": ">1.0.2 <=2.3.4",
  "boo": "2.0.1",
}
```

Ukázka kódu 5: Package.json - závislosti projektu

Závislost je definována jménem balíčku a verzí, ve které má být využíván. Jméno je řetězec totožný s pojmenováním balíčku na npmjs.com. Verze je definována číslem.[25]

5.3 Význam čísel ve verzích

Díky možnostem zápisu vyžadované verze je možné vyjádřit velice konkrétně, za jakých podmínek může koncový uživatel používat jinou verzi balíčku, než s jakou byl projekt původně vyvíjen. s tímto záměrem na mysli je nutné vědět, co jednotlivé číslice znamenají.[25]

5.3.1 1.0.0 - První vydání

Verze 1.0.0 je doporučovaným výchozím čísle verze nového balíčku. verze 1.0.0 tedy typicky značí zcela nový balíček bez žádných dalších aktualizací.[25]

5.3.2 1.0.1 - Oprava chyby, zpětně kompatibilní

Inkrementace třetí číslice značí opravu chyby z předchozí verze. Důležitou podmínkou je, že tato aktualizace nijak neovlivní zpětnou kompatibilitu se staršími verzemi balíčku. Pokud tedy využijeme zápis vyžadované verze typu 1.0.x (viz strana 39), umožníme uživateli využívat novější verze balíčku s opravenými

chybami bez obav ze změny funkčnosti oproti verzi se kterou byl projekt vyvíjen.[25]

5.3.3 1.1.0 - Nová funkce, zpětně kompatibilní

Inkrementace druhé číslice značí přidání nové funkce, či vlastnosti balíčku. i v tomto případě je stěžejní fakt, že přidání této funkce nijak neovlivňuje zpětnou kompatibilitu s předchozími verzemi balíčku se stejnou první číslicí.[25]

Při inkrementaci druhé číslice se resetuje hodnota na třetí pozici zpět na nulu. Pokud je tedy poslední verze 1.1.7 a je přidána nová funkce, číslo nové verze bude 1.2.0.[25]

5.3.4 2.0.0 - Nová funkce, bez zpětné kompatibility

Inkrementací první číslice se značí zásadní změna ve funkčnosti balíčku. Může to znamenat přidání nových funkcí, které nejsou zpětně kompatibilní, odebrání dříve existující funkcí, či jiné zásadní přepracování balíčku.[25]

Pro nové verze této úrovně je typické, že aktualizace z předchozích verzí vyžaduje zásah vývojáře do projektů, kde je tento balíček používán, neboť je pravděpodobné že způsob jakým je balíček využíván potřeba upravit. Při změně první číslice se druhá a třetí pozici resetuje na nulu.[25]

5.4 Zápis požadované verze balíčku

Pokud při definici závislosti uvádíme pouze konkrétní číslo verze balíčku, bude balíček vyžadován vždy v této konkrétní verzi. Kromě toho máme k dispozici více variant pro zápis požadované verze balíčku.[25]

5.4.1 * - hvězdička

Hodnota závislosti vyjádřená hvězdičkou značí, že balíček může být využit v jakékoliv jeho verzi. Což může znamenat, že uživatel není nucen stahovat konkrétní verzi balíčku, který už v jiné verzi ve svém systému má. Stejný

efekt má i situace, kdy požadovanou verzi neuvedeme a přítomny jsou pouze uvozovky.[25]

5.4.2 >1.0.0 - větší než

Zápis s tímto operátorem jednoduše značí, že projekt vyžaduje přítomnost balíčku ve verzi novější než je verze 1.0.0. Tento zápis je používán například v případě, kdy určitá, projektem využívaná, funkce balíčku je přítomna právě až od verzí s číslem vyšší než 1.0.0. Také jinak, verze 1.0.0 je poslední, která není dostačující.[25]

5.4.3 >=1.0.0 - větší nebo rovno

Zápis větší nebo rovno se může vzhledem k existenci výše zmíněného zápisu jevit jako zbytečný, ale není tomu tak. Představme si situaci, kdy poslední verze balíčku, jež využíváme jsou verze číslo 1.9.0 a 2.0.0. v našem projektu chceme využít nové funkce, jež je přítomna právě od verze 2.0.0. Mohlo by se zdát, že vhodným zápisem tedy bude >1.9.0, to ale nemusí nutně znamenat verzi 2.0.0. Je možné, že i po vydání verze 2.0.0 přibude např. verze s číslem 1.9.5, která nebude obsahovat změny verze 2.0.0. Tato zpětná podpora může existovat z toho důvodu, že pro účely některých projektů nemusí být vždy nejvhodnější aktuální verze, neboť je možné že právě s příchodem verze 2.0.0 byli odstraněny některé funkce, které byli považovány autorem za zbytečné, ale přesto jsou pro jiné tvůrce stěžejní.[25]

V takovém případě může po odhalení chyby autor vydat novou verzi s opravou verze 1.9.0 a tedy verzi 1.9.5. Tím bude splněna podmínka přítomnosti verze novější než 1.9.0, ale můžou nastat problémy, jelikož vývojář počítal s minimální verzí 2.0.0.[25]

5.4.4 <1.0.0 - menší než, <=1.0.0 - menší nebo rovno

Obdobně můžeme vyjádřit i požadovanou verzi s číslem menším, či menším nebo rovnou uvedenému číslu. Platí zde stejná logika a pravidla jako u zápisu „větší“ a „větší než“.[25]

5.4.5 1.1.x

Symbol „x“ v tomto kontextu zastupuje typickou funkci hvězdičky (*). Zápis znamená, že je vyžadována verze, kde první dvě číslovky jsou 1.1 a poslední může být libovolná. To znamená, že je možné využití verzí 1.1.1, 1.1.2 atd., ale už nikoliv verzí 1.2.0 a vyšších.[25]

Stejně tak můžeme použít zástupný symbol na více místech např. 1.x.x čímž vymezíme přijatelné verze v rozmezí od 1.0.0 do 2.0.0 kde první nepřijatelnou verzí tedy bude právě verze 2.0.0.[25]

5.4.6 <1.9.0 || >=2.0.0

Požadovanou verzi také můžeme definovat pomocí rozsahů. Zápis <1.9.0 || >=2.0.0 znamená, že požadovaná verze může být s číslem menším než 1.9.0 nebo s číslem větším, než 2.0.0. Tento zápis můžeme použít např. v případě, kdy jsme si vědomi, že mimo tento rozsah obsahuje balíček chybu, jež omezuje funkčnost našeho projektu.[25]

5.5 Package-lock.json

Package-lock.json je generovaný soubor, jež je přítomen v hlavní složce projektu po boku package.json. Jeho existence má několik účelů.[26]

- Popisuje konkrétní reprezentaci závislostí projektu, v konkrétním bodu v čase. Umožňuje všem vývojářům (např. kolegům v týmu) instalaci stejných verzí jednotlivých závislostí.

- Umožňuje uživateli „vrátit se v čase“, ke konkrétnímu stavu závislostí.
- Pomáhá optimalizovat instalační proces tím, že umožní přeskakovat, již existující položky dříve nainstalovaných balíčků.[26, 27]

Hlavní rozdíl package-lock.json oproti package.json je následující. Package.json definuje přepis pro závislosti a jejich požadované verze, oproti tomu package-lock.json popisuje reálný, současný stav nainstalovaných balíčků a jejich verzí. Package.json tedy sami vytváříme na základě balíčků, které chceme využívat a verzí, které jsou přijatelné. Package-lock.json se generuje dle reálného stavu a na rozdíl od package.json, by neměl být ručně editován.[27]

Pokud si projekt stáhne další osoba např. za účely vývoje, můžeme se tak vyvarovat situaci, kdy rozdíl ve verzi balíčku způsobuje u dvou vývojářů rozdílné chování projektu.[27]

Možnost „vrácení v čase“ do minulého stavu nainstalovaných balíčků, se může hodit v situaci, kdy po aktualizaci balíčků nastane problém s funkčností projektu. Díky informaci, v jakém verzi byli balíčky v minulosti používány, můžeme projekt vrátit do funkčního stavu a ujistit se, že problém s funkčností je opravdu chyba některé z aktualizovaných balíčků.[27]

Zápis uvnitř souboru package-lock.json se na první pohled mírně liší od zápisu v souboru package.json.[27]

```
"@firebase/polyfill": {
  "version": "0.3.3",
  "resolved": "https://registry.npmjs.org/@firebase/
     polyfill/-/polyfill-0.3.3.tgz",
  "integrity": "sha512-xs8IZf1WEbufYXyfV8YjmiFZOaujRRq0
     T03NteihYfuGVTTym7z5SmvLvEHLEUjf2fgeobPEzZ2JgrCQHS
     +QHw==",
  "requires": {
      "core-js": "2.5.5",
      "promise-polyfill": "7.1.2",
      "whatwg-fetch": "2.0.4"
  },
  "dependencies": {
      "core-js": {
    "version": "2.5.5",
    "resolved":"http://registry.npmjs.org/core-js/-/
       core-js-2.5.5.tgz",
    "integrity": "sha1-sU3ek2xkDAV5prUMq8wTLdYSfjs=",
    },
    "whatwg-fetch": {
    "version": "2.0.4",
    "resolved":"http://registry.npmjs.org/whatwg-fetch
       /-/wha...",
    "integrity":"sha512-dcQ1GWpOD/eEQ97k66aiEVpNnapVj90
       /+R+SX..."
    }
  }
},
```

Ukázka kódu 6: Package-lock.json - balíček @firebase/polyfill

Zde můžeme vidět záznam z package-lock.json pro balíček polyfilu projektu Firebase. Některé hodnoty mají stejné označení jako v souboru package.json, ale jejich význam se může lišit.[27]

5.5.1 Version

Položka „version“ přesně udává verzi, ve které je balíček přítomen.[26]

5.5.2 Resolved

Položka „resolved“ obsahuje adresu, z které se tato verze stahovala. Jedná se o kompletní URL adresu, jež typicky směřuje do registru balíčků NPM.[26]

5.5.3 Integrity

Bezpečnostní kód, jež ověřuje stáhnutí balíčku z dané adresy bez cizí manipulace.[26]

5.5.4 Dev

Položka „dev“ může nabývat hodnoty true, nebo false. Hodnota true znamená, že balíček je součástí vývojářských devDependencies.[26]

5.5.5 Optional

Nabývá hodnot true, nebo false. Hodnota true značí, že instalace balíčku je volitelná. Znamená tedy, že balíček není nutné instalovat pro funkčnost projektu, či pro běh vývojářského procesu. Může se jednat např. o balíčky, jež jsou kompatibilní jen s jedním operačním systémem. Takový balíček může na konkrétním operačním systému usnadňovat práci na projektu, a přesto nebýt povinnou součástí pro vývojáře, či uživatele na jiných platformách.[26]

5.5.6 Requires

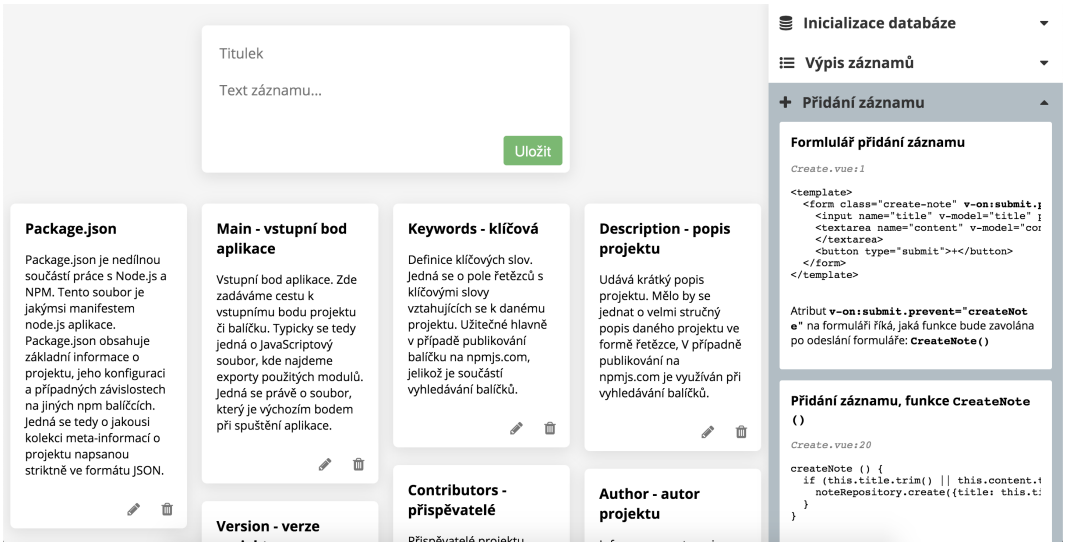
Jedná se o závislosti definované v souboru package.json. Jedná se o seznam všeho, co balíček vyžaduje bez ohledu na to, co bude reálně nainstalováno.[26]

5.5.7 Dependencies

Reálně nainstalované závislosti balíčku. Tedy jaké balíčky byly nainstalovány za účelem správně funkce tohoto balíčku.[26]

6 Praktická část

Praktickým výstupem této práce je single-page webová aplikace. Tato aplikace umožňuje uživatelům přidávat textové záznamy do databáze, které jsou posléze vypisovány přímo do jejího rozhraní. Všem uživatelům, je zobrazována stejná kolekce záznamů. Záznam obsahuje titulek, textový obsah, jméno autora a datum přidání. Uživatelé mohou záznamy libovolně upravovat, či mazat. Uživatel také může záznamy filtrovat a seřadit dle jejich autora, či data přidání. Všechny zápisy do databáze jsou v reálném čase synchronizovány napříč výpisy záznamů všech aktivních uživatelů.



Obrázek 1: Praktický výstup - single-page aplikace

Významnou částí aplikace, je průvodce jejími funkcemi. Při zobrazení aplikace v desktopovém rozlišení můžeme najít v pravé části obrazovky seznam popisující funkce aplikace. Tento průvodce není zaměřen na funkčnost z pohledu uživatele aplikace, ale na fungování aplikace z pohledu vývojáře. Kompletní zdrojový kód aplikace je dostupný jako veřejný repozitář, dostupný z: `http://github.com/sverakjan/sverak-notes-app`. Průvodce je poté doplňkem ke zdrojovému kódu. Průvodce je rozdělen do logických prvků pojednávajících o konkrétní funkční části aplikace, jako například mazání záznamu.

Obrázek 2: Průvodce funkcemi aplikace

Zde se můžeme dozvědět jaký kód je využíván k realizaci této funkcionality, najít jeho vysvětlení a v neposlední řadě získat informaci v jakém souboru a na jakém řádku tento kód v aplikaci najdeme.

Hlavním účelem této aplikace není konkurovat existujícím produktům podobné povahy, nýbrž přestavovat ukázku Node.js aplikace s podobnými funkcemi. Má sloužit jako demonstrace, či studijní materiál pro další vývojáře, jež chtějí reálnou ukázku tvorby Node.js aplikací.

Aplikace je dostupná z adresy `http://sverak-notes-app.firebaseapp.com/`.

6.1 Použité přístupy a technologie

Při tvorbě tohoto projektu, byli kromě Node.js a NPM balíčků, využity i další technologie, jež jeho tvorbu znatelně usnadnili a přinesli další užitečné funkce.

6.1.1 Single-page aplikace

Praktickým výstupem této práce, je takzvaná single-page aplikace (SPA), to s sebou z technického hlediska přináší několik rozdílů oproti klasickým webovým stránkám.[28]

Webové stránky se tradičně považují za kolekci HTML souborů, pro jednotlivé stránky doplněné o obrázky, CSS, JS atd. Při načtení stránky server poskytne zařízení klienta HTML soubor s obsahem stránky a při přechodu na jinou část webu je mu poskytnut další HTML soubor s obsahem nové stránky. Pro SPA je typické, že celý web je tvořen (alespoň co se HTML týče), pouze jedním souborem. Odtud pochází označení single-page aplikace, a tedy aplikace obsahující pouze jednu stránku. Tato jediná stránka neobsahuje samotný obsah webu, namísto toho zde, kromě nezbytných částí HTML souborů a odkazů na externí zdroje, typicky najdeme HTML značku:[28]

```
<app></app>
```

Tato značka není součástí specifikace HTML5, přesto je často využívaná při tvorbě single-page aplikací. Jak bylo zmíněno ve vstupním HTML souboru SPA není obsah, ten je dynamicky dodáván namísto značky <app></app>. Pokud chce uživatel přejít na jinou část webu, potažmo aplikace, není přesměrován na další HTML soubor, nýbrž pouze obsah stránky je aktualizován tak, aby odpovídal cílené části. Díky tomu musí, zařízení koncového uživatele, při přechodu stahovat pouze tu část obsahu, která se změní. Ty části stránky, jež zůstávají stejné nemusí uživatel při přechodu na jinou část obsahu webu znovu stahovat, což prohlížení webu, či používání aplikace zrychluje. k dynamickému dodávání obsahu do stránky se využívá JavaScript.[29]

Tento přístup k tvorbě webových aplikací se hodí například v případě, kdy uživateli chceme zobrazit často se měnící data a opakované kompletní načítání stránky by přineslo zbytečnou datovou zátěž a negativně by ovlivnilo uživatelský zážitek z používání aplikace.[28]

6.1.2 Vue.js

Za účelem usnadnění tvorby single-page aplikací, jsou často využívané frameworky, jež tuto práci značně urychlují. Pro účely zpracování praktické části práce, byl využit framework Vue.js.[28]

Vue.js je JavaScriptový framework určený pro tvorbu uživatelských rozhraní. Jedná se tedy o frontendový framework, který usnadňuje tvorbu single-page webových aplikací.[30]

Stejně jako v případě dále zmíněného projektu Firebase je tento framework přístupný uživatelům Node.js jako NPM balíček, jeho instalace tedy není nijak náročná.

6.1.3 Firebase

Firebase je projekt vlastněný společností Google. Skládá se z několika nástrojů a služeb, jako například cloudové úložiště a hosting webových aplikací, nástroje pro autentizaci uživatelů, nástroje pro testování, či NoSQL cloudovou databázi. Právě tato databáze je využita pro ukládání a organizaci dat v projektu praktické části této práce. Jak již bylo zmíněno, konkrétně se jedná o „Firebase Realtime Database“. Hlavní výhody použití NoSQL databáze jsou popsané v kapitole číslo 3.5.[10, 31]

Další výhodou je, že aplikace využívající této databáze, může zůstat aktivní i v případě, že zařízení, na kterém aplikace běží, ztratí připojení k internetu. Tohoto je docíleno udržováním využívaných dat přímo na zařízení uživatele. Ve chvíli, kdy je připojení k internetu opět navázáno, zařízení získá změny,

o které přišlo za dobu, kdy bylo off-line a je tím synchronizováno se současným stavem databáze.[10]

V kontextu této práce se tedy můžeme vyvarovat nepříjemnostem spojeným s nestabilním připojením uživatele. Pokud uživatel například pracuje ve vlaku pomocí mobilního připojení k internetu, nemusí se bát, že aplikace při výpadku z důvodu chvilkového ztracení signálu přestane fungovat. v situaci, kdy v době výpadku napíše nový záznam a pokusí se jej odeslat, vidí vše proběhnout v pořádku a nový záznam vidí ve výpisu jako běžně. Rozdíl je pouze v tom, že tento záznam není uložen v databázi, přístupné ostatním uživatelům, ale pouze lokálně u uživatele, jenž jej vytvořil. Ve chvíli, kdy je připojení k internetu obnoveno, je tato změna ihned odeslána a zobrazena i ostatním uživatelům. i přes výpadek nebude v očích uživatele funkčnost aplikace nijak ovlivněna a jeho uživatelský zážitek je tak nezměněn, i při využívání nestabilního připojení k internetu.[10]

Klíčovou funkcí použité Firebase Realtime Database je její synchronizace dat v reálném čase. Namísto typických HTTP požadavků využívá tato databáze rychlé synchronizace dat pokaždé, když dojde záznamech databáze k nějaké změně. To znamená, že po každém zápisu do databáze je aktualizovaná její verze pro každé připojené zařízení v řádu milisekund.[10]

V případě této práce tak můžeme dosáhnout kýženého efektu, kdy záznam přidán kterýmkoliv uživatelem je takřka okamžitě viditelný i pro všechny ostatní připojené uživatele.[10]

6.2 Inicializace databáze

Jak již bylo zmíněno, v práci je využito projektu Firebase. Firebase je součástí NPM jako balíček, proto můžeme Firebase jednoduše přidat do projektu pomocí příkazu:

```
npm install firebase --save
```

Poté již lze s Firebase v projektu dále pracovat. Pro lepší přehlednost kódu je pro základní práci s databází vytvořen soubor Firebase.js.

Prvním krokem je importování samotné Firebase pomocí příkazu:

```
import * as firebase from "firebase";
```

Dalším krokem je definování konstanty, jež obsahuje konfigurační údaje pro samotnou databázi:

```
1 const config = {
2   apiKey: "AIzaSyDhvdDaXVBc25RVdZA4DKdMurqZUd1ejUE",
3   authDomain: "sverak-notes-app.firebaseapp.com",
4   databaseURL: "https://sverak-notes-app.firebaseio.com
      ",
5   projectId: "sverak-notes-app",
6   storageBucket: "sverak-notes-app.appspot.com",
7   messagingSenderId: "715631935704"
8 };
```

Ukázka kódu 7: Konstanta pro konfiguraci databáze

Tyto údaje můžeme získat po registraci a vytvoření projektu na stránkách Firebase určených pro vývojáře: `console.firebase.google.com`. v administraci projektu je nám tento kód rovnou poskytnut na základě vyplněných informací, jako například jméno projektu.

Nejdůležitější položka při inicializaci je „apiKey“. Bez vygenerovaného klíče není možné přistupovat k Firebase API.

Posledním krokem inicializace databáze je exportování. k tomu slouží kód:

```
1 export default !firebase.apps.length ?
2 firebase.initializeApp(config) : firebase.app();
```

Ukázka kódu 8: Export inicializace databáze

Tento export nám umožňuje přístup k databázi, bez nutnosti inicializace v každém souboru, kde k téže databázi přistupujeme a tím zabraňujeme redundantnosti a možným chybám s tím spojeným.

6.3 Výpis záznamů

Abychom mohli uživateli vypsat záznamy musíme mít přistup k databázi, kde jsou uložené. Díky výše zmíněné konfiguraci a exportu můžeme nyní k databázi jednoduše přistupovat.

K tomu slouží kód, jež najdeme v souboru Node Repository.js:

```
1 constructor () {
2   super()
3   var database = firebase.database();
4 }
```

Ukázka kódu 9: Konstruktor pro Firebase databázi

V našem případě, budou všechny zápisy do databáze a čtení z databáze na větvi „notes“. Pro usnadnění přístupu je tedy opět s oboru NoteRepository.js připravena funkce:

```
1 get notesRef () {
2   return firebase.database().ref(`notes`);
3 }
```

Ukázka kódu 10: Getter pro Firebase databázi

Výchozím bodem vykreslování aplikace pomocí Vue.js je soubor Index.vue. Pro výpis záznamů jsou tedy v souboru Index.vue přítomny tyto dva důležité importy:

```
import Note from './Note'
import noteRepository from '../../data/NoteRepository'
```

Ukázka kódu 11: Import markupu a funkcí pro záznam

V souboru Note.vue mimo jiné najdeme markup pro samotný záznam. v souboru NoteRepository.js získáváme přístup k samotné databázi a událostem jako přidání, aktualizace a smazání záznamu.

Jednotlivé záznamy jsou přístupné jako pole a u každého záznamu máme přístup ke třem hodnotám. Note.key (jedinečný identifikátor záznamu), note.title (text titulku) a note.content (samotný text záznamu). Při samotném výpisu záznamu na stránky využijeme poslední dvě zmíněné.

HTML Markup záznamu zapsaný pomocí syntaxe Vue.js v souboru Note.vue vypadá takto:

```
<template>
  <div class="note" v-bind:class="[size]">
    <h2>{{note.title}}</h2>
    <pre>{{note.content}}</pre>
    <button type="button" v-on:click.stop="remove">
      <i class="fa fa-trash-o" aria-hidden="true"></i>
    </button>
    <button class="edit" type="button">
      <i class="fa fa-pencil" aria-hidden="true"></i>
    </button>
  </div>
</template>
```

Ukázka kódu 12: Markup pro výpis záznamu

Markup výpisu jednotlivých záznamů je statický, jedinou proměnnou složkou je text titulku a obsahu záznamu. Tyto hodnoty jsou reprezentovány jako note.title a note.content.

6.4 Přidání záznamu

Vytváření nových záznamů je uskutečněno pomocí formuláře. Jeho markup najdeme v souboru Create.vue:

```
<template>
  <form class="create-note"    v-on:submit.prevent="
     createNote">
    <input name="title" v-model="title" placeholder="
       Titulek"/>
    <textarea name="content" v-model="content"
       placeholder="Text zaznamu..." rows="3">
    </textarea>
    <button type="submit">+</button>
  </form>
</template>
```

Ukázka kódu 13: Markup formuláře pro přidání záznamu

Atribut v-on:submit.prevent="createNote" na elementu formuláře říká, jaká funkce bude volána po odeslání formuláře. Jedná se o funkci CreateNote(), kterou taktéž najdeme v souboru Create.vue. v zjednodušené podobně vypadá funkce takto:

```
createNote () {
  if (this.title.trim() || this.content.trim()) {
    noteRepository.create({title: this.title, content:
       this.content})
  }
}
```

Ukázka kódu 14: Funkce createNote - zjednodušená verze

Tento kód by nebylo zcela vhodné využít v reálné aplikaci. Využita je tedy komplexnější varianta s ošetření chyby a vypisováním upozornění pro uživatele:

```
createNote () {
  if (this.title.trim() || this.content.trim()) {
    noteRepository.create({title: this.title, content:
       this.content}, (err) => {
    if (err) return this.$dispatch('alert', {type: '
       error', message: Zaznam se nepodarilo vytvorit'}
       )
      this.title = ''
      this.content = ''
      this.$dispatch('alert', {type: 'success', message
         : 'Zaznam vytvoren'})
    })
  }
}
```

Ukázka kódu 15: Funkce createNote - komplexnější varianta

Funkce this.title.trim() odstraní z řetězce mezery a zalomení řádků. Podmínka tedy bude splněna pokud v hodnotě titulku je nějaký znak, který není whitespace (například mezera, či ukončení řádku).

Tato funkce je použita pouze pro kontrolu při tvorbě záznamu. Záznam je poté uložen a vypisován včetně těchto znaků. Vypisované záznamy jsou tedy i včetně mezer a zalomených řádků.

Pokud je tedy tato podmínka splněna, může být posléze volána funkce noteRepository.create(), které předáváme hodnotu title (Titulek) a content (Text záznamu) z elementů input a textarea.

Kód samotné funkce noteRepository.create() můžeme najít v souboru NoteRepository.js:

```
create ({title = '', content = ''}, onComplete) {
  this.notesRef.push({title, content}, onComplete)
}
```

Ukázka kódu 16: Funkce create - přidání záznamu do databáze

NotesRef je pole obsahující všechny záznamy. Funkce tedy vkládá položku do pole.

Tato funkce odpovídá Firebase metodě určené pro vkládání položek do databáze dle specifikace dokumentace.[32]

6.5 Úprava záznamu

Uživatel má možnost upravit existující záznam. Kliknutí na záznam vyvolá modální okno s formulářem. Markup tohoto modálního okna najdeme v souboru UpdateModal.vue:

```
1  <template>
2    <div v-if="note" transition="modal" class="backdrop"
       v-on:click="dismissModal">
3      <form class="edit-note" v-on:submit.prevent="update
         " v-on:click.stop="">
4        <input name="title" v-model="note.title"
           placeholder="Titulek"/>
5        <textarea name="content" v-model="note.content"
           placeholder="Text zaznamu..." rows="8"></
           textarea>
6        <button type="button" v-on:click="remove">
7        <i class="fa fa-trash-o" aria-hidden="true"></i>
8        </button>
9        <button type="submit">Hotovo</button>
10     </form>
11   </div>
12 </template>
```

Ukázka kódu 17: Markup modalu pro úpravu záznamu

Atribut v-on:submit.prevent="update" na elementu formuláře říká, jaká funkce bude volána po odeslání formuláře. v tomto případě se jedná o funkci update(), kterou opět najdeme v souboru UpdateModel.vue:

```
update () {
  noteRepository.update(this.note, (err) => {
    if (err) return this.$dispatch('alert', {type: '
       error', message: 'Nepodarilo se zaznam
       aktualizovat'})
    this.dismissModal()
    this.$dispatch('alert', {type: 'success', message:
       'Zaznam byl upraven'})
  })
}
```

Ukázka kódu 18: Funkce update - úprava záznamu

Funkce update() ve skutečnosti volá funkci ze souboru noteRepository a záznam na něž bylo kliknuto předává funkci jako parametr. Ne element záznamu, jaký je vykreslený na stránce pro uživatele, ale hodnotu „key“ tedy jednoznačný identifikátor konkrétního záznamu. Společně s touto hodnotou jsou funkci předané i nové texty zadané ve formuláři modalu.

V případě, že funkce noteRepository.update() vrátí chybu, tato funkce uživateli zobrazí výstražnou hlášku. Pokud se tak nestane, znamená to, že záznam byl úspěšně upraven a uživateli je zobrazena hláška oznamující úspěch operace.

Jak bylo zmíněno, za samotnou úpravu záznamu v databázi je zodpovědná funkce NoteRepository.update(), jež najdeme v souboru NoteRepository.js:

```
update ({key, title = '', content = ''}, onComplete) {
  this.notesRef.child(key).update({title, content},
     onComplete)
}
```

Ukázka kódu 19: Funkce update - aktualizace záznamu v databázi

Funkce na základě dodaného klíče najde položku v poli záznamů a pokud byla poskytnuta nová hodnota pro titulek, aktualizuje tuto hodnotu přímo v poli databáze. To samé je provedeno pro hodnotu obsahu záznamu.

To má za výsledek aktualizace záznamu v databázi, ale na stránce zobrazené uživateli bude stále zobrazena původní neupravená verze záznamu. k tomu slouží kód, který najdeme v souboru Index.vue:

```
noteRepository.on('changed', ({key, title, content}) =>
    {
  let note = noteRepository.find(this.notes, key)
  note.title = title
  note.content = content
})
```

Ukázka kódu 20: Aktualizace záznamu na vykreslené stránce

Poté co je záznam aktualizován v poli databáze, je vyvolána událost, jež aktualizuje reprezentaci záznamu, kterou vidí na stránce uživatel. Událost je vyvolána automaticky při změně v databázi. Cílem této funkce je pouze synchronizovat obsah stránky, kterou vidí uživatel s aktuálním stavem databáze.

6.6 Mazání záznamu

Uživatel vidí na každém záznamu symbol popelnice. Při kliknutí na tuto ikonku dojde k odstranění záznamu. v kódu se jedná o běžné tlačítko:

```
<button type="button" v-on:click.stop="remove">
```

Ukázka kódu 21: Tlačítko pro odstranění záznamu

Atribut v-on:submit.prevent="remove" na tlačítku udává, jaké funkce bude volána při kliknutí na tlačítko zobrazovaném jako ikona popelnice. Jedná se o funkci remove(), kterou najdeme v souboru Note.vue:

```
remove () {
  noteRepository.remove(this.note, (err) => {
    if (err) return this.$dispatch('alert', {type: '
        error', message: 'Nepodarilo se odstranit zaznam
        })
  })
}
```

Ukázka kódu 22: Funkce remove - odstranění poznámky

Obdobně jako funkce update() i funkce remove() volá funkci ze souboru noteRepository.js a záznamu na něž bylo kliknuto předává funkci jako parametr, opět ne samotný element záznamu jaký je vykreslený na stránce, ale hodnotu „key“ tedy jednoznačný identifikátor konkrétního záznamu. v případě, že funkce noteRepository.remove() vrátí chybu, tato funkce uživateli zobrazí výstražnou hlášku.

Volanou funkci NoteReposiotry.remove(), již přirozeně najdeme v souboru NoteRepository:

```
remove ({key}, onComplete) {
  this.notesRef.child(key).remove(onComplete)
}
```

Ukázka kódu 23: Funkce remove - odstranění záznamu z databáze

Tato funkce pouze odstraní položku z pole obsahující všechny záznamy. Tím odstraní zápis záznamu z databáze, ale pro uživatele bude na stránce stále viditelný. k tomu slouží funkce v souboru Index.vue:

```
noteRepository.on('removed', ({key}) => {
  let note = noteRepository.find(this.notes, key)
  this.notes.$remove(note)
})
```

Ukázka kódu 24: Odstranění záznamu na vykreslené stránce

Poté co je záznam odstraněn z pole databáze, je vyvolána událost, jež odstraní reprezentaci záznamu zobrazeného na stránce. v případě odstranění záznamu viditelné uživatelem se jedná o prosté smazání elementu ze stránky.

7 Závěr

Bakalářská práce se zabývá využitím jazyka JavaScript na straně serveru, jež je možné díky frameworku Node.js. V teoretické části byli představeny obecné informace spojené s frameworkem Node.js a jeho balíčkovacím systému NPM. Dále bylo porovnáno využití Node.js a NoSQL databáze s tradičně využívanými technologiemi PHP a MySQL.

V praktické části byla vytvořena demonstrativní single-page webová aplikace, jejímž hlavním účelem je představovat funkční ukázku Node.js projektu. Přítomnost průvodce jejími funkcemi z této aplikace dělá praktický studijní materiál pro vývojáře, jež tvorba Node.js aplikací zajímá.

Možnost využití jazyka JavaScript při tvorbě serverové části webu, či webové aplikace otevírá možnost podílet se na tomto procesu i jedincům, jež se primárně orientují v technologiích určených pro klientskou část webu. Využití JavaScriptu na obou stranách vývoje může pomoci přiblížit tyto dvě tradičně oddělené oblasti a tím nám pomoci k snazší komunikaci a spolupráci frontendových a backedových vývojářů.

I přesto nemůžeme tvrdit, že většina nových webů a webových aplikací bude tvořena s pomocí této technologie. Node.js není vhodný pro všechny typy projektů a v těchto případech je stále na místě využití jiné technologie jako je PHP. Obdobně můžeme mluvit o využití NoSQL databází a tradičně využívané MySQL databázi. Obecné porovnání těchto technologií je obtížné, jelikož Node.js a PHP nebyly vyvinuty se stejným záměrem. i přesto můžeme říct, že pro konkrétní případy využití přináší Node.js oproti PHP řadu výhod.

Single-page aplikace, jež byla vytvořena v praktické části je typickým příkladem projektu, kde je výhodné využít Node.js. Jedná se zejména o aplikace, kde je stěžejním zpracování dat v reálném čase a práce s velkým množstvím, rychle se měnících dat. Můžeme předpokládat, že poptávka po aplikacích takové povahy bude nadále růst a s ní i popularita Node.js.

Seznam použité literatury a zdrojů

[1] JOYENT, INC. About Node.js [online]. [cit. 2019-04-02]. Dostupné z: <https://nodejs.org/en/about/>

[2] W3C. Node.js Introduction [online]. [cit. 2019-04-02]. Dostupné z: <https://www.w3schools.com/nodejs/nodejs_intro.asp>

[3] GOOGle, INC. What is V8? [online]. [cit. 2019-04-02]. Dostupné z: <https://developers.google.com/v8/>

[4] YOUTEAM. Top Compatnies that Used Node.js in Production [online]. 22. 1. 2018 [cit. 2019-04-02]. Dostupné z: <https://youteam.co.uk/blog/top-companies-that-used-node-js-in-production/>

[5] JOYENT, INC. How do i start with Node.js after i installed it? [online]. [cit. 2019-04-02]. Dostupné z: <https://nodejs.org/en/docs/guides/getting-started-guide/>

[6] HAYES, David B. Why Use PHP in 2019? [online]. 12. 6. 2018 [cit. 2019-04-02]. Dostupné z: <https://www.thoughtfulcode.com/why-use-php/>

[7] UPWORK GLOBAL INC. PHP vs JavaScript [online]. [cit. 2019-04-02]. Dostupné z: <https://www.upwork.com/hiring/development/php-vs-javascript/>

[8] RAMBHIYA, Vishal. What is the difference between PHP and Node.js? [online]. 8. 7. 2018 [cit. 2019-04-11]. Dostupné z: <https://www.citacepro.com/dok/Uvk747CimMpWlboE>

[9] RACHOWICZ, Justyna. When, How And Why Use Node.js as Your backend [online]. 23. 2. 2017 [cit. 2019-04-02]. Dostupné z: <https://www.netguru.com/blog/use-node-js-backend>

[10] GOOGLE, INC. Firebase Realtime Database [online]. [cit. 2019-04-02]. Dostupné z: <`https://firebase.google.com/docs/`>

[11] KOPAL, Ondřej. Relační a nerelační datový model v kontextu business intelligence [online]. 16. 11. 2015 [cit. 2019-04-11]. Dostupné z: <`http://www.web-integration.info/cs/blog/relacni-a-nerelacni-datovy-model-v-kontextu-business-intelligence/`>

[12] XPLENTY, The SQL vs NoSQL Difference: MySQL vs MongoDB [online]. 28. 9. 2017 [cit. 2019-04-02]. Dostupné z: <`https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2`>

[13] MONGODB INC. What is NoSQL [online]. [cit. 2019-04-02]. Dostupné z: <`https://www.mongodb.com/nosql-explained`>

[14] NPM, INC. About npm [online]. [cit. 2019-04-02]. Dostupné z: <`https://docs.npmjs.com/about-npm/index.html`>

[15] ROGERS, Mikeal. Request - npm [online]. [cit. 2019-04-02]. Dostupné z: <`https://www.npmjs.com/package/request`>

[16] KOVALYOV, Antom. JSHint, a Static Code Analysis Tool for JavaScript - npm [online]. [cit. 2019-04-02]. Dostupné z: <`https://www.npmjs.com/package/jshint`>

[17] HARTL, Klaus. JavaScript Cooke - npm [online]. [cit. 2019-04-02]. Dostupné z: <`https://www.npmjs.com/package/js-cookie`>

[18] NPM, INC. About packages and modules [online]. [cit. 2019-04-02]. Dostupné z: <`https://docs.npmjs.com/about-packages-and-modules`>

[19] TAL, Liran. terrified of NPM security? please don't blindly follow the panic [online]. 8. 1. 2015 [cit. 2019-

04-02]. Dostupné z: <https://blog.cloudboost.io/npm-security-please-dont-blindly-follow-the-panic-f724871ba1a8>

[20] HÁMORI, Ferenc. Controlling the Node.js security risk of npm dependencies [online]. [cit. 2019-04-02]. Dostupné z: <https://blog.risingstack.com/controlling-node-js-security-risk-npm-dependencies/>

[21] GITHUB INC. About security alerts for vunerable dependencies [online]. [cit. 2019-04-02]. Dostupné z: <https://help.github.com/en/articles/about-security-alerts-for-vulnerable-dependencies>

[22] NPM, INC. Npm-nistall, install a package [online]. [cit. 2019-04-02]. Dostupné z: <https://docs.npmjs.com/cli/install>

[23] CYREN, Tierney. The Basic of Package.json in Node.js and npm [online]. 3. 3. 2017 [cit. 2019-04-02]. Dostupné z: <https://nodesource.com/blog/the-basics-of-package-json-in-node-js-and-npm/>

[24] MICHÁLEK, Martin. Package.json: Vývojářský manifest pro každý projekt [online]. 4. 11. 2018 [cit. 2019-04-02]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/package-json>

[25] NPM, INC. Npm-package.json Specific of npm's package.json handling [online]. [cit. 2019-04-02]. Dostupné z: <https://docs.npmjs.com/files/package.json>

[26] NPM, INC. Npm-package-lock.json a manifestation of the manifest [online]. [cit. 2019-04-02]. Dostupné z: <https://nodesource.com/blog/the-basics-of-package-json-in-node-js-and-npm/>

[27] MICHÁLEK, Martin. Package-lock.json: Proč potřebujeme "lockfile"? [online]. 12. 11. 2018 [cit. 2019-04-02]. Dostupné z: <https://www.vzhurudolu.cz/prirucka/package-lock-json>

[28] GOOGLE, INC. Angular Single Page Applications (SPA): What are the Benefits? [online]. [cit. 2019-04-02]. Dostupné z: <https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/>

[29] QUALITY NONSENCE LTD. <app> HTML Tag [online]. [cit. 2019-04-02]. Dostupné z: <https://html.com/tags/app/>

[30] YOU, Evan. What is Vue.js? [online]. [cit. 2019-04-02]. Dostupné z: <https://vuejs.org/v2/guide/>

[31] GOOGLE, INC. Firebase documentation [online]. [cit. 2019-04-02]. Dostupné z: <https://firebase.google.com/docs/>

[32] GOOGLE, INC. Firebase Realtime Database - Saving Data [online]. [cit. 2019-04-02]. Dostupné z: <https://firebase.google.com/docs/database/admin/save-data>

Seznam obrázků

Seznam ukázek kódu

8 Přílohy

1. CD se zdrojovými kódy celé aplikace a plné znění BP v PDF

2. Webová stránka: `https://sverak-notes-app.firebaseapp.com/`