Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Architektura webových aplikací

Bakalářská práce

Autor: Vojtěch Kycelt
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové Duben 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 18.4.2023 Vojtěch Kycelt

Poděkování:

V první řadě děkuji svému vedoucímu doc. Ing. Filipu Malému, Ph.D. za vstřícné jednání a odborné vedení při psaní bakalářské práce. Dále bych rád poděkoval Ing. Adamu Černohorskému za příležitost podílet se na vývoji rezervačního systému Online Termín, která mě uvedla do světa webových technologií.

Anotace

Cílem této bakalářské práce jsme zvolili získání přehledu o využívaných technologiích při tvorbě webových aplikací, definování vhodné architektury pro konkrétní případy a seznámení budoucích vývojářů i ostatních čtenářů s danou problematikou.

V první části seznámíme čtenáře s pojmy týkajícími se webových aplikací a historií jejich architektur. Po vysvětlení pojmu webové aplikace se přesuneme k základní struktuře celé webové aplikace a konkrétním technologiím pro její vývoj.

Následující částí představíme webový framework a jeho funkci jakožto základního kamene webové aplikace.

V závěru práce pak využijeme znalosti získané v předešlých kapitolách, použijeme je k vytvoření aplikací z různých technologií a následně porovnáme jejich efektivitu v různých situacích.

Annotation

Title: Web application architecture

The aim of this bachelor's thesis is to gain an overview of the technologies used during the creation of web applications, to define a suitable architecture for specific cases and to familiarize future developers and other readers with the given issue.

In the first part, the reader is introduced to concepts related to web applications and the history of their architectures. After explaining the concept of a web application, we will move on to the basic structure of the entire web application and specific technologies for its development.

In the following section we will present web framework and its function as the cornerstone of a web application.

At the end of the thesis, we will use the knowledge gained in the previous chapters to create applications using different technologies and then compare their effectiveness in different situations.

Obsah

1 Úvod

Již před několika lety začaly webové aplikace pronikat do všech moderních společností. Dnes už těžko budeme hledat oblast, do které ještě nepronikly, a často se v různých případech bez kvalitní webové aplikace ani neobejdeme. V oblasti webových aplikací zažívá internet neustálý rozvoj. Tento rozvoj sebou přináší stále vyšší a vyšší nároky na webové aplikace, a proto se jejich architektura stále mění a vyvíjí. Objevují se stále nové trendy, jiné naopak zase zanikají.

V této práci představíme moderní koncept architektury webové aplikace a ukážeme, jak je možné tento koncept využít ve dvou různých aplikacích. V první aplikaci využijeme architekturu typu MERN a druhou aplikaci vytvoříme z kombinace technologií VueJS, Spring Boot a PostgreSQL. Obě aplikace složíme ze tří hlavních vrstev, jejichž vrchní části budou tvořeny javascriptovým front-endem, kterým zajistíme dynamiku webových stránek. V back-endových částech aplikací porovnáme využití JavaScriptu a Javy, a následně spodními částmi jednotlivých aplikací porovnáme rozdíly mezi dokumentovou a objektově-relační databází.

V MERN aplikaci si ukážeme využití technologie ReactJS. Další část o vrstvu níže vyplníme server-side frameworkem s názvem ExpressJS, který poběží uvnitř NodeJS serveru. Touto kombinací dodáme aplikaci efektivní nástroje pro směrování URL a ošetřování HTTP požadavků. Většina aplikací potřebuje ukládat data, která nezmizí po zavření stránky v prohlížeči, a o to se postaráme databázovou částí, neboli poslední vrstvou této architektury. MongoDB, jako databázová technologie, dokáže vynikajícím způsobem ukládat data v JSON dokumentech, které jsou vytvářeny v ReactJS front-endu a zpracovávány pomocí ExpressJS serveru.

V druhé aplikaci na front-endové části využijeme framework VueJS, který v back-endové části doplníme Spring Bootem. Touto kombinací přineseme bližší pohled na propojení vývoje javascriptového front-endu s back-endem psaným v Javě. V databázové vrstvě využijeme technologii PostgreSQL a ukážeme rozdílné vlastnosti objektově-relační databáze oproti dokumentové databázi.

V rámci hlavního cíle této práce představíme proces tvorby webové aplikace, vysvětlíme funkčnost takové architektury a podrobněji seznámíme s výše zmíněnými technologiemi. Jako další cíl ukážeme účinné využití těchto technologií při tvorbě webových aplikací a v poslední části použijeme informace získané z praktické části projektu k porovnání a efektivnímu návrhu technologií pro různé případy užití.

Pro vytvoření této práce využijeme zkušeností z tvorby webových aplikací získáných během mého dvouletého působení ve firmě Exorior systems s.r.o., kde působím jako front-endový vývojář různých aplikací převážně psaných v technologii ReactJS.

2 Webová aplikace

Termín webová aplikace je podle autorů (SHKLAR & ROSEN, 2003) něco víc než jen "webová stránka". Tento software funguje jako aplikace typu klient/server, která využívá webový prohlížeč jako klientský program, a serverová část se nachází na internetu nebo intranetu. Tyto aplikace poskytují interaktivní služby prostřednictvím komunikace mezi klientem a serverem.

Takové aplikace často kombinují HTML, CSS a JavaScript s jinými jazyky, jako je například PHP, Python nebo Ruby, aby poskytly dynamický a interaktivní obsah. Tyto aplikace v interaktivní formě jsou například e-shopy nebo sociální sítě, dále mohou poskytovat informace jako například encyklopedie nebo novinové stránky.

Oproti webové aplikaci je webová stránka pouhý statický dokument nebo sada dokumentů, které jsou dostupné na internetu a obsahují text, obrázky, videa, odkazy a další prvky. Rozdílem mezi webovou stránkou a webovou aplikací je tedy to, že webová aplikace poskytuje uživateli možnost interakce, zatímco webová stránka je jen statický dokument.

2.1 Historie

Zpočátku lidé na internetu sdíleli převážně statické informace v souborech. Tyto soubory mohly být upravovány a jejich obsah aktualizován, ale neexistovalo dost skutečně dynamických informačních služeb a na internetu jich byl nedostatek (SHKLAR & ROSEN, 2003).

Existovalo jen několik výjimek jako aplikace pro vyhledávání souborů v archivech FTP a Gopheru nebo služby, které přímo poskytovaly dynamické informace, jako například počasí nebo dostupnost plechovek z automatu na limonádu. Většinou však byly informační zdroje sdílené na webu statické dokumenty (SHKLAR & ROSEN, 2003).

Vše změnil příchod dynamických informačních služeb, které zahrnovaly škálu od vyhledávačů přes CGI skripty až po balíčky, které propojovaly web s relačními databázemi. S příchodem dynamického webu se laťka zvedla ještě výše. Už nestačilo říct, že navrhujeme "webovky" jako kolekci webových stránek. Bylo nutné navrhnout webovou aplikaci. Jedna z prvních webových aplikací, které Tim Berners-Lee předvedl

v CERNu, byla brána pro vyhledávání na webu z databáze telefonního seznamu pomocí webového prohlížeče (SHKLAR & ROSEN, 2003).

2.1.1 První aplikace a jejich vývoj

Historie vývoje webových aplikací je poměrně zajímavá a neobvyklá. Vývojáři museli hledat co nejradikálnější a nejintenzivnější řešení pro existující problémy. Bylo důležité zajistit, aby webové aplikace fungovaly hladce na různých operačních systémech.

První počítačové modely byly nevyhovující. Každá aplikace měla svůj vlastní předkompilovaný klientský program, který musel být nainstalován na každém počítači uživatele zvlášť. Komponenty klienta a serveru byly navíc pevně spojené s určitým operačním systémem a architekturou počítače. V důsledku toho bylo nákladné přenášet aplikace na jiné systémy.

Ve fázi webu v jeho raných dnech, o které hovoří (MALÍK, 2019), si uživatelé webu vystačili se statickými stránkami a jednoduchými uživatelskými rozhraními. Nicméně při práci s těmito prvními webovými stránkami bylo těžké získat interaktivní zážitek, a tak s rozvojem nových programovacích jazyků, webových prohlížečů, mobilních zařízení a výkonnějších procesorů, vzrostla očekávání uživatelů ohledně interakce a použitelnosti webových aplikací. Tyto požadavky vedly k vývoji nové technologie a rozšíření možností webových prohlížečů. V popředí vešly moderní architektury aplikací ovlivněné přístupy k návrhu jako je REST nebo GraphQL, jejichž cílem je určitým způsobem usnadnit komunikaci serveru s klientem.

Rok 1995 je zásadním rokem v éře internetu. Společnost Netscape Communications Corporation se snažila vytvořit programovací jazyk, který by byl více přístupný než psaní v Javě a zároveň vytvořit v Netscape Navigatoru podporu pro Javu, která by byla snazší pro programátory neprogramující v Javě (HOLZNER, 2003). Tato společnost dosáhla svého cíle a 4. prosince téhož roku představila JavaScript, skriptovací jazyk na straně klienta umožňující programátorům přidávat do uživatelského rozhraní dynamické prvky (HOLZNER, 2003). Díky JavaScriptu se internet stal rychlejším a produktivnějším, protože data se již neposílala na server, aby se vygenerovala celá webová stránka. Vložené skripty plní různé úkoly na konkrétní stažené stránce "přímo na místě". JavaScript je jednou ze tří nejvýznamnějších technologií (spolu s HTML a CSS)

tvorby obsahu pro WWW. Disponuje aplikačním programovým rozhraním, které odborníkům umožňuje pracovat s texty, daty a různými regulárními výrazy. Ve skutečnosti však nedisponuje vstupem/výstupem, díky němuž by mohl "komunikovat" s okolním světem.

V roce 1996 byla uvedena na trh aplikace Macromedia Flash, která představovala další revoluční novinku, jež přinesla do webového prostředí interaktivitu a větší barevnost. Tato multimediální platforma umožňovala tvorbu animací, různých typů her pro prohlížeče, vektorové grafiky a internetových a mobilních aplikací. Adobe Flash, jenž zahrnul do své animace i možnost streamování zvuku a videa, znamenal solidní posun vpřed v této oblasti. Tato platforma umožňovala uživateli komunikovat se strojem pomocí myši, mikrofonu a klávesnice. Od roku 1996 byla zaznamenána rostoucí popularita různých interaktivních online videoher díky revoluční technologii, kterou poskytla společnost Macromedia Flash, avšak v následujících letech popularita zase poklesla. Práci uživatele přerušovaly podivné a nečekané reklamy a streamovaná videa zpomalovávající práci webu a spotřebovávající další provoz. Přesto do nedávna existovaly levné webové stránky, které Flash ve svých aplikacích využívaly.

V uplynulých letech byl Adobe Flash především používán pro tvorbu videoher a interaktivních aplikací pro chytré telefony a tablety. Nicméně, od července 2021 Microsoft úplně zrušil podporu pro Adobe Flash přehrávač a trvale odstranil tuto komponentu ze svých prohlížečů.

Roku 1999 se objevil koncept webové aplikace v jazyce Java, který v roce 2005 následovalo představení techniky zvané Ajax. Tato technika, kterou popsal Jesse James Garrett ve svém článku "Ajax: A New Approach to Web Application", umožnila programátorům tvorbu asynchronních webových aplikací. Hotové aplikace umožňují uživatelům pracovat na webu rychleji a efektivněji, protože mohou odesílat a získávat data na serveru bez potřeby načítat celou stránku. Původně byla vytvořena pro Internet Explorer, ale později byla převzata i jinými prohlížeči, jako jsou Opera, Mozilla a Safari. Společnost Google úspěšně používala tuto techniku v aplikacích Gmail a Google Maps od roku 2005.

Podle (MASON, 2020) se v průběhu let objevilo mnoho verzí jazyka HTML. HTML5, nejnovější verze jazyka HTML, spatřilo světlo světa v roce 2014. Je určeno k prezentaci obsahu na WWW a jeho uspořádání do logických struktur. Tento jazyk byl vytvořen jako vylepšení stávajícího standardu HTML a jeho úkolem je podporovat zcela nové typy multimédií, které se stále vyvíjejí. HTML5 sám o sobě není schopen poskytovat animace ani streamovaná videa, ty se musí zprostředkovat pomocí JavaScriptu. Podle průzkumu alespoň 34% nejpopulárnějších webových stránek používá HTML5, což naznačuje jeho vysokou důležitost v oblasti webového vývoje.

V roce 2015 Alex Russel a Frances Berman představili moderní koncept zvaný progresivní webové aplikace (PWA). Tyto webové aplikace jsou navržené tak, aby využívaly nejnovějších technologií a nabízely uživatelům stejnou funkčnost jako nativní aplikace. PWA se tedy snaží "vzít si všechny správné vitamíny" a nabídnout uživatelům lepší zážitek a funkčnost (ROJAS, 2019).

Interaktivita webu se stala obrovskou a má vysoké předpoklady pro to, stát se ještě efektivnější a rozmanitější. Ajax představuje jeden z nejlepších příkladů metod, které zlepšují úroveň interaktivity mezi uživatelem a strojem. Bezpochyby budeme svědky rychlého zdokonalování internetových technologií, zejména webových aplikací. Nejchytřejší lidé tohoto oboru se snaží o to, aby se webové aplikace staly nezávislými na platformách, prohlížečích a jiných aplikacích.

2.1.2 Dnešní aplikace

V dnešních aplikacích byla logika přesunuta ze serveru na klienta, což umožnilo zvýšenou interakci na straně klienta bez nutnosti neustálé komunikace se serverem. Myš jako vstupní zařízení také sehrála důležitou roli v evoluci aplikací. Pomocí ní byli uživatelé schopni manipulovat s netextovými prvky obrazovky, což se uplatnilo hlavně u programů pracujících s multimédii (např. editory obrázků či videí) (MALÍK, 2019). To, co nebylo možné s terminálovými aplikacemi, se stalo skutečným s příchodem bohatých klientů. Až doposud bylo nereálné si představit např. úpravu obrázků ve webové aplikaci, protože by to vyžadovalo neustálé znovunačítání stránek při každé operaci. Avšak moderní webové aplikace využívají zpracování i na straně klienta, čímž odlehčují server od práce a zároveň zvyšují interaktivitu (MALÍK, 2019).

3 Struktura webové aplikace

Na internetu vysoce převažují aplikace typu klient-server, o kterých hovoří (SHKLAR & ROSEN, 2003). Tyto aplikace jsou souborem distribuovaných programů, které běží na vzdálených počítačích a komunikují mezi sebou pomocí specifických protokolů. Místo toho, aby veškeré zpracování probíhalo v jediném systému a výsledky byly zobrazovány na jednoduchých terminálech, aplikace klient-server rozdělují zpracování mezi vyhrazené terminály, jako jsou servery a klientské počítače. Tuto architekturu umožnilo rozšíření osobních počítačů, které poskytly dostatečný výpočetní výkon pro přesunutí některých úkonů ze serverů na klienty.

3.1 Klient

Webová aplikace typu klient-server je aplikace, která obecně využívá webový prohlížeč jako klienta. Prohlížeče posílají požadavky serverům a servery generují požadavky a odpovědi a vrací je prohlížečům. Od starších systémů klient-server se liší tím, že využívají společný klientský program, konkrétně prohlížeč (SHKLAR & ROSEN, 2003).

Použití webových prohlížečů jako klientů přináší značné výhody. V první řadě webové prohlížeče jsou součástí většiny operačních systémů, tudíž jsou přítomny prakticky na každém stolním počítači, notebooku, tabletu či mobilním telefonu. Tyto klienty se dají snadno použít k interakci s mnoha jinými aplikacemi přítomnými na webu a není třeba instalovat další specializované klientské programy na používaný stroj, což má za následek snížení problémů s údržbou a výkonem. Dále mají prohlížeče zabudované mechanismy pro bezpečné stahování a spouštění jiných klientů, jako například appletů, pokud je vyžadována funkce, kterou konkrétní prohlížeče poskytnout nemohou (SHKLAR & ROSEN, 2003).

3.1.1 HTML

Pod zkratkou HTML se skrývá název HyperText Markup Language. Nejedná se o programovací jazyk v tradičním slova smyslu. Dokumenty tohoto značkovacího jazyka, které jsou základem veškerého obsahu na WWW, se skládají ze dvou základních částí. První část zahrnuje informační obsah a druhou částí je soubor instrukcí, které počítači říkají, jak má informační obsah zobrazit. Webový prohlížeč jakožto klient si překládá

soubor instrukcí a obsah zobrazí. V ideálním případě by měl online obsah vypadat stejně bez ohledu na prohlížeč nebo na operační systém, ve kterém je prohlížeč umístěn. Tohoto cíle úplné nezávislosti na platformě je dosaženo v praxi jen přibližně (BROOKS, 2014).

Základní dokument HTML potřebuje nejméně čtyři sady prvků, které ho definují. Mezi tyto sady patří samotný dokument, nadpis, hlavička a tělo. Každý z těchto prvků je definován dvěma značkami, a to počáteční a koncovou. Značky, nebo také tagy, jsou vždy uzavřeny v hranatých závorkách a koncové značky začínají lomítkem. Ačkoliv většina tagů by se měla vyskytovat v párech, některé prvky HTML mají značku pouze jednu a párové pravidlo je v jazyce HTML dodržováno jen volně (BROOKS, 2014).

```
<html> ... </html>
<head> ... </head>
<title> ... </title>
<body> ... </body>
```

Pro podporu skriptovacího jazyka, jako je JavaScript, je nutno přiložit ke čtyřem základním prvkům další specifický prvek *<script>*. Tento prvek vždy obsahuje kód JavaScriptu. Značka *<html>* uzavírá všechny ostatní značky a vymezuje hranice dokumentu HTML. Značky scriptů se často objevují uvnitř hlavičky (*<head>*), ale mohou se objevit i jinde v dokumentu. Odsazení slouží k vyčlenění dvojic značek, je nepovinné, ale usnadňuje vytváření, čtení a úpravy dokumentů. Tento styl je součástí správné programovací praxe ve všech jazycích (BROOKS, 2014).

Ačkoli jsou dokumenty HTML obvykle považovány za způsob distribuce informací pro vzdálený přístup na webu, jsou stejně užitečné i při lokálním použití na jakémkoli počítači, který má prohlížeč. Ve spojení s jazykem JavaScript tedy můžete vytvořit samostatné prostředí pro řešení problémů, které lze používat jak lokálně, tak i globálně (BROOKS, 2014).

3.1.2 CSS

Kaskádové styly neboli CSS, jsou jazykem, který udává prezentaci dokumentu. CSS lze použít s většinou značkovacích jazyků včetně XUL a SVG a prakticky s jakýmkoli dokumentem XML podporujícím styly. Jazyk CSS se již před několika lety stal základní

součástí webového zásobníku a stejně jako jiné technologie představuje několik výzev v oblasti zabezpečení (HEIDERICH, NAVA, HEYES & LINDSAY, 2011).

CSS umožňuje vývojářům oddělit logiku webové stránky od jejího vzhledu, což usnadňuje úpravy a opakované použití stylu na různých stránkách. Tento jazyk obsahuje mnoho různých možností pro stylování prvků, jako jsou například barvy, velikosti písma, okraje a mezery. Kromě toho CSS nabízí možnosti pro pozicování prvků na stránce, jako jsou absolutní nebo relativní pozice. Díky CSS lze také snadno přizpůsobit vzhled webové stránky pro různé velikosti obrazovek, například pro mobilní zařízení nebo pro tablety. Mezi další funkce tohoto jazyka patří vytváření animací a transformací, které pomáhají zvýšit interaktivitu webových aplikací.

3.1.3 JavaScript

JavaScript je interpretovaný, multiplatformní, objektově orientovaný programovací jazyk, jehož syntaxe vychází z jazyka Java a byl vyvinut pro použití spolu s dalšími webovými nástroji. JavaScript nefunguje jako samostatný jazyk, ale je navržen tak, aby spolupracoval s jazykem HTML a umožňoval vytváření interaktivních webových aplikací (BROOKS, 2014).

Vzhledem k tomu, že dokumenty HTML/JavaScript jsou pouze textové dokumenty, mohou být vytvářeny pomocí libovolného textového editoru. Pro jednoduché úlohy je použitelná dokonce i základní aplikace Poznámkový blok systému Windows. Po jejich vytvoření lze otevřít soubory HTML v prohlížeči počítače (BROOKS, 2007).

JavaScript není stejný jako Java, která je kompilovaným objektově orientovaným jazykem. JavaScript je oproti Javě velmi uvolněný jazyk, kde nemusí být deklarovány všechny proměnné, třídy a metody. Není nutné se starat o to, zda jsou třídy veřejné, soukromé nebo chráněné, ani implementovat různá rozhraní. Navíc mizí potřeba explicitně určovat typy proměnných, parametrů a návratových hodnot funkcí (JavaScript, 2022), pro tuto možnost lze použít TypeScript, neboli JavaScript rozšířený o kontrolu nad typy.

JavaScript se používá k psaní aplikací na straně klienta, což znamená, že kód v jazyce JavaScript je odeslán do počítače uživatele při načtení webové stránky. Tento kód

je pak prováděn v podstatě řádek po řádku interpretem JavaScriptu, který je součástí webové stránky prohlížeče uživatele. Tímto uspořádáním jsou minimalizovány problémy s bezpečností, které se mohou objevit při komunikaci klientského počítače s počítačem, který stránku odeslal. Umožňuje také snadno zabalit celou problematiku s vlastním uživatelským rozhraním a řešením, samostatně obsaženým v jediném dokumentu. Nemožnost dynamické interakce s informacemi uloženými na serveru však omezuje druhy úloh, které může JavaScript řešit (BROOKS, 2014).

Je běžné označovat jakýkoli soubor písemných počítačových instrukcí jako "program". Tento termín se však přísněji používá pro samostatnou entitu, kterou lze spustit samostatně. Vzhledem k tomu, že JavaScript je interpretovaný, nikoliv kompilovaný, samostatně spustitelná entita se nikdy nevytváří. Místo toho jsou příkazy kódu JavaScriptu interpretovány a prováděny jeden po druhém, v podstatě "za běhu". Ačkoli se to může zdát neefektivní, jen zřídka dochází k nějaké znatelné časové prodlevě spojené s prováděním příkazů jazyka JavaScript na moderních počítačích (BROOKS, 2014).

JavaScript patří do třídy skriptovacích jazyků, jejichž účelem je přistupovat k součástem existujícího informačního rozhraní a upravovat je. V tomto případě je tímto rozhraním dokument HTML. Jakmile se dokumenty HTML na webu vyvinuly z jednosměrných doručovacích systémů pro zobrazování pevného obsahu, stal se JavaScript okamžitě nezbytným. Jedna z jeho prvních aplikací vyplynula z potřeby kontrolovat hodnoty zadané uživateli do polí formulářů HTML, které mohou být odeslány zpět původci. JavaScript může použít k porovnání vstupních hodnot s očekávaným rozsahem nebo sadou hodnot a k vygenerování na základě těchto porovnání příslušné zprávy a další akce (BROOKS, 2014).

3.2 Server

Server je základní součástí webové aplikace klient/server. Je to počítač nebo skupina počítačů, které jsou vybaveny softwarem a hardwarem pro poskytování služeb klientům. Hlavní úlohou serveru je poskytovat informace a funkce klientům, kteří se k němu připojují prostřednictvím sítě.

Webové servery umožňují přístup HTTP k "webové stránce", což je jednoduše kolekce webových dokumentů a dalších informací uspořádaných do stromové struktury podobně jako souborový systém počítače. Kromě přístupu ke statickým dokumentům, servery také implementují řadu protokolů pro předávání požadavků vlastnímu softwaru (BROOKS, 2014).

Webové servery, prohlížeče a proxy servery mezi sebou komunikují prostřednictvím výměny zpráv HTTP. Server přijímá a interpretuje požadavky HTTP, vyhledává a zpřístupňuje požadované zdroje a generuje odpovědi, které posílá zpět původcům požadavků (BROOKS, 2014).

3.2.1 NodeJS

NodeJS je asynchronní běhové prostředí JavaScriptu řízené událostmi, které je určeno k vytváření škálovatelných síťových aplikací. Lze zpracovávat mnoho spojení současně. Při každém spojení se spustí zpětné volání, ale pokud není třeba provést žádnou práci, Node se uspí.

Node je svým designem podobný systémům, jako je Event Machine v jazyce Ruby nebo Twisted v jazyce Python, a je jimi ovlivněn. Node jde v modelu událostí ještě o něco dál a představuje smyčku událostí jako běhovou konstrukci namísto knihovny. V jiných systémech je vždy k dispozici blokující volání pro spuštění smyčky událostí. Typicky je chování definováno prostřednictvím zpětných volání na začátku skriptu a na konci je spuštěn server přes blokující volání. V NodeJS takové volání pro spuštění smyčky událostí neexistuje. Node jednoduše vstoupí do smyčky událostí po provedení vstupního skriptu a ukončí smyčku událostí, když už není třeba provádět žádná další zpětná volání. Toto chování se podobá JavaScriptu v prohlížeči, smyčka událostí je před uživatelem skryta (Node JS, 2023).

NodeJS je sice navržen bez vláken, ale dovoluje využít více jader ve svém prostředí. Procesy potomků lze spouštět pomocí API volání a jsou navrženy tak, aby se s nimi dalo snadno komunikovat. Na stejném rozhraní je postaven modul cluster, který umožňuje sdílet sockety mezi procesy a umožnit tak rozložení zátěže na jádra (Node JS, 2023).

3.3 Databáze

Databáze je software nebo systém, který ukládá a umožňuje přístup k datům. Existují různé typy databází, jako například relační databáze, dokumentové databáze, grafové databáze a key-value databáze. Relační databáze jsou nejrozšířenější typ databází a používají relační model pro ukládání dat v tabulkách s řádky a sloupci. Na druhé straně dokumentové databáze ukládají data ve formátu dokumentů, jako jsou JSON nebo XML. Grafové databáze slouží k ukládání a vyhledávání dat v grafové struktuře a key-value databáze ukládají data jako páry klíč-hodnota.

3.3.1 MongoDB

MongoDB je oblíbená databáze typu NoSQL, která se od klasických relačních databází odlišuje dokumentovým modelem. Data jsou ukládána v dokumentech, které jsou podobné formátu JSON, a následně organizována do kolekcí, což je obdobné relačním tabulkám. Na rozdíl od tradičních databází v MongoDB není potřeba předem definovat schéma a tabulky, což umožňuje rychlejší vývoj a flexibilitu vůči změnám datových struktur (MongoDB, 2023).

Jednou z hlavních výhod MongoDB je jeho schopnost pracovat s velkými objemy dat. Dokumentový model umožňuje jednodušší rozdělení dat na různé kolekce a sharding, což umožňuje rozšířit kapacitu databáze na více strojů. MongoDB také podporuje automatickou replikaci, což zvyšuje dostupnost a odolnost databáze. Další výhodou MongoDB je jeho flexibilní databázové schéma. U relačních databází je nutné předem definovat schéma tabulek, což může být omezující pro aplikace, které potřebují pracovat s různými typy dat. MongoDB umožňuje ukládat různé typy dat do stejné kolekce, což přináší snadnou integraci a rozšíření dat pro aplikace (MongoDB, 2023).

MongoDB je multiplatformní a podporuje různé programovací jazyky, jako jsou JavaScript, Python, C++, Java a další. Existuje také široká komunita kolem MongoDB, která poskytuje podporu a další nástroje pro práci s touto databází (MongoDB, 2023).

MongoDB se často používá pro aplikace, které potřebují pracovat s velkými objemy dat, jako jsou například internetové obchody, sociální sítě nebo aplikace pro správu velkých

datových souborů. Navzdory svému využití pro velké projekty, MongoDB může být také používán pro menší aplikace (MongoDB, 2023).

3.3.2 PostgreSQL

PostgreSQL nebo zkráceně Postgres spadá do objektově-relačních databázových systémů. Vysokou výkonnost doplňuje otevřený zdrojový kód, na kterém se vývojáři aktivně podílí už 35 let. Tato technologie získala velmi dobrou pověst díky své spolehlivosti, robustnosti funkcí a výkonu (PostgreSQL, 2023).

Postgres disponuje mnohými funkcemi jako jsou například databázové transakce s vlastnostmi ACID neboli atomicitou, konzistencí, izolací a trvanlivostí. Dále nabízí aktualizované a materializované pohledy, triggery, cizí klíče a uložené procedury. Tento databázový systém byl navržen tak, aby zvládal širokou škálu zátěže, od jednotlivých strojů až po datové sklady nebo webové služby s více uživateli, a umožňuje tak snadné zpracování rozsáhlých datových objemů.

Postgres je vynikající volbou pro aplikace, které potřebují ukládat a spravovat velké množství dat a je vybaven mnoha funkcemi, které umožňují ukládání a manipulaci s daty na vysoké úrovni. Podporuje mnoho různých typů dat a umožňuje ukládání vztahů mezi nimi pomocí klíčů cizích klíčů. Postgres je také známý pro svou schopnost efektivně zpracovávat složité dotazy (VOHRA, 2016).

Postgres se používá především pro aplikace, které pracují s relačními daty, jako jsou například bankovní aplikace nebo systémy pro správu skladu.

4 Webový framework

Frameworky neboli rámce pro vývoj webových aplikací označují soubor zdrojů a nástrojů, které jsou k dispozici vývojářům softwaru a webových aplikací. Tyto frameworky poskytují vývojářům webových aplikací možnost vytvářet a spravovat webové aplikace, webové služby a webové stránky.

Jedná se o softwarový rámec, který byl vyvinut s cílem zjednodušit proces vývoje webových stránek a usnadnit jejich tvorbu (PELZER, 2019). Obsahuje šablonovací funkce, které umožňují prezentovat informace v prohlížeči, poskytuje prostředí pro skriptování toku informací a obsahuje také mnoho aplikačních programových rozhraní (API) pro získání přístupu k základním datovým zdrojům. Většina frameworků také poskytuje nástroje pomáhající vývojářům webových stránek vytvářet systémy správy obsahu (CMS) pro správu digitálních informací na webových stránkách a internetu.

Vývojový framework, jak o něm hovoří (PELZER, 2019), lze považovat za předem vytvořenou strukturu, která zvládá opakující se procesy a funkce spojené s vývojem webových stránek. To znamená, že vývojář webu stráví většinu času interakcí s různými částmi webového frameworku pomocí kódu.

Mezi funkce spojené s webovými frameworky patří například ukládání do mezipaměti webu, postupy ověřování a autorizace, mapování a konfigurace databáze a mapování adres URL.

4.1 Typy webových frameworků

Programátoři mají k dispozici mnoho různých druhů této platformy. Typicky, jako spoustu věcí v rámci tohoto oboru, lze rozdělit na front-endový a back-endový vývoj.

4.1.1 Back-endové frameworky

Back-endové frameworky automatizují mnoho funkcí a procesů, které vývojáři stránek využívají k záznamu a načítání dat zadaných uživatelem. Obecně jsou tyto frameworky navrženy pro usnadnění práce s databází.

ExpressJS

ExpressJS je framework pro NodeJS, který poskytuje uživatelům jednoduchý a flexibilní způsob vytváření webových aplikací a API. Díky své popularitě se stává jednou z nejčastěji používaných knihoven pro tvorbu webových aplikací na platformě NodeJS (Express JS, 2023).

Express se zaměřuje na snadnost použití a minimalismus, což umožňuje rychlé a efektivní vytváření webových aplikací. Nabízí řadu funkcí, jako jsou routování, middleware a řízení požadavků a odpovědí, které umožňují snadnou implementaci běžných funkcí webových aplikací. Express také podporuje integraci s jinými knihovnami a frameworky, jako jsou MongoDB, Socket.io nebo PassportJS. To umožňuje snadnou implementaci funkcí, jako je autentizace nebo komunikace v reálném čase. Express navíc podporuje asynchronní práci, což umožňuje efektivně využívat výkon platformy NodeJS. To dovoluje aplikaci práci s více požadavky současně a minimalizuje čas odezvy pro uživatele (Express JS, 2023).

V poslední době se Express stává stále populárnějším a používá se mnoha velkými společnostmi, jako jsou IBM, Accenture nebo Uber. Je to jedna z nejpopulárnějších knihoven pro vývoj webových aplikací na platformě NodeJS a je považována za jednu z nejlepších voleb pro vývoj moderních webových aplikací.

Spring

Spring Framework je open-source Java framework, který se zaměřuje na snadný vývoj aplikací. Byl vytvořen Rodem Johnsonem v roce 2002 a je nyní vyvíjen komunitou open-source (Spring, 2023).

Spring se zaměřuje na snadný vývoj aplikací s vysokou mírou struktury a organizace. Nabízí širokou škálu modulů, které pomáhají vývojářům vytvořit aplikace s vysokou mírou flexibility a škálovatelnosti. Spring dodává silnou podporu pro Dependency Injection (DI) a Inversion of Control (IoC), což umožňuje snadnou organizaci a reutilizaci kódu. Spring nabízí silnou podporu pro práci s databázemi, webovými službami a různými druhy úložišť. Dále přináší rozsáhlou knihovnu "Spring Boot" pro automatizaci práce s projektem a pomáhá vývojářům vytvářet aplikace rychleji a efektivněji (Spring, 2023).

Spring Framework je jedním z nejpopulárnějších frameworků pro vývoj Java aplikací. Je používán v mnoha velkých společnostech jako IBM, VMWare nebo Netflix a je považován za jednu z nejlepších voleb pro vývoj moderních Java aplikací.

4.1.2 Front-endové frameworky

Front-endové frameworky jsou velmi užitečné pro prezentační stránku webu a vytvářejí dobrý základ, na kterém lze stránky stavět. Tyto frameworky kombinují jazyky HTML, CSS a JavaScript a jsou skutečně účinné při vytváření konvencí pro různé prvky. Obvykle je vývojáři front-endových webů používají ve spojení s dalšími účinnými nástroji.

Konvence HTML umožňují standardizaci prvků, jako je například typografie, zatímco konvence CSS pomáhají standardizovat umístění prvků v systému podobné mřížce na stránkách. To vývojářům usnadňuje práci uspořádáním a organizací prvků do jednoduchého a unifikovaného designu napříč různými zobrazeními a prohlížeči. Konvence jazyka JavaScript nabízejí možnost stylovat pokročilejší komponenty, jako jsou obrázkové karusely, chování tlačítek a vyskakovací okna.

Existují frameworky JavaScriptu, které se zaměřují na plnou aplikační podporu, jako například Angular. Při vývoji konvencí, které jsou snadno implementovatelé do webových stránek, pomáhají také knihovny kódu Javascript, jako jsou jQuery a ReactJS.

Používání těchto nástrojů pomáhá vývojáři rychle začít pracovat, aniž by se musel zdržovat počátečními překážkami, které s sebou přináší tvorba webových aplikací. Vytváří jednotný kód na všech stránkách webu a představuje pro uživatele čistý vzhled. To zase řeší běžné problémy, jako je konzistence typografie a potíže s umístěním na různých stránkách. Další skvělou vlastností front-endových frameworků je skutečnost, že jsou schopny řešit veškeré problémy s kompatibilitou prohlížečů a jsou klíčem k responzivnímu webovému designu.

ReactJS

ReactJS je javascriptová knihovna sloužící k tvorbě uživatelských rozhraní. Byla vytvořena firmou Facebook a je nyní vyvíjena jako open-source projekt. ReactJS je charakterizován jako deklarativní, efektivní a flexibilní a zaměřuje se na vytváření jednotlivých komponent, které se dají snadno kombinovat do složitějších aplikací. Tyto komponenty jsou jednoduše plněné daty a automaticky se aktualizují, pokud se data změní (React JS, 2023).

React také nabízí funkci "virtual DOM", která umožňuje efektivní a rychlou aktualizaci rozhraní uživatele. Virtual DOM je kopie skutečného DOMu, která se používá pro vyhodnocení změn, které se mají provést. Tím se minimalizuje počet operací, které jsou nutné k aktualizaci rozhraní (React JS, 2023).

React se často používá spolu s dalšími technologiemi, jako je Redux nebo MobX, pro správu stavu aplikace. Tyto knihovny poskytují jednoduchý způsob, jak udržovat stav aplikace centrálně a umožňují snadnou komunikaci mezi různými komponentami (React JS, 2023).

V poslední době se React stává stále populárnějším a je používán mnoha velkými společnostmi, jako je Facebook, Airbnb nebo Netflix. Je to jedna z nejpopulárnějších knihoven pro vývoj uživatelských rozhraní a je považován za jednu z nejlepších voleb pro vývoj moderních webových aplikací.

VueJS

VueJS je open-source javascriptový framework pro vývoj uživatelských rozhraní. Byl vytvořen Evanem Youem v roce 2014 a je nyní vyvíjen komunitou open-source. Vue se zaměřuje na snadnost použití a flexibilitu, což umožňuje vývojářům rychle a snadno vytvářet komplexní uživatelská rozhraní (Vue JS, 2023).

Vue nabízí komponentový model, který umožňuje snadnou organizaci a reutilizaci kódu a vytváří "reaktivní systém", který automaticky aktualizuje uživatelské rozhraní, pokud se data změní. Tento systém umožňuje vývojářům snadno sledovat a upravovat data v aplikaci. Vue dále podporuje "Single-File Components" (SFC), což dovoluje vývojářům kombinovat HTML, CSS a JavaScript do jednoho souboru. To umožňuje snadnou organizaci kódu a snižuje počet souborů, které je potřeba spravovat (Vue JS, 2023).

V poslední době se Vue stává stále populárnějším a používá se mnoha společnostmi jako Alibaba nebo Xiaomi. Je to jedna z nejpopulárnějších knihoven pro vývoj uživatelských rozhraní a je považován za jednu z nejlepších voleb pro tvorbu moderních webových aplikací (Vue JS, 2023).

Angular

Angular je bezplatný framework vyvinutý společností Google, který využívá jazyka JavaScript a zaměřuje se na vytváření komplexních aplikací s vysokou mírou struktury a organizace. Nabízí komponentový model, který umožňuje snadnou organizaci a reutilizaci kódu. Angular využívá "Two-Way Data Binding", což umožňuje automatickou synchronizaci dat mezi modelem a uživatelským rozhraním. Dále nabízí rozsáhlou knihovnu "Angular CLI" pro automatizaci práce s projektem a pomáhá vývojářům vytvářet aplikace rychleji a efektivněji. Angular podporuje TypeScript, který poskytuje silné typování a pomáhá snižovat chyby v kódu. Angular dále vyniká silnou podporou pro testování, což umožňuje vývojářům snadno testovat jednotlivé části aplikace a udržovat vysokou kvalitu kódu. Angular také nabízí rozsáhlou dokumentaci a komunitu podpory, což přináší vývojářům snadné nalezení odpovědí na jejich otázky (Angular, 2023).

První verze Angularu byla vydána v roce 2010 a od té doby se stal jedním z nejpopulárnějších frameworků pro vývoj webových aplikací. Používá se v mnoha velkých společnostech jako Google, IBM nebo Microsoft a je jedním z nejpopulárnějších frameworků pro vývoj webových aplikací. Je považován za jednu z nejlepších voleb pro vývoj moderních webových aplikací s vysokou mírou struktury a organizace (Angular, 2023).

5 Praktická část

Praktickou částí této práce si vysvětlíme postup vytvoření full-stack webových aplikací. První aplikací zvolíme MERN neboli stack technologií MongoDB, ExpressJS, React a NodeJS. Druhou aplikaci vytvoříme z technologií VueJS, Spring Boot a PostgreSQL. Při implementaci jednotlivých aplikací a jejich určitých částí si budeme ukazovat různá řešení i to, pro jaké případy je jejich použití vhodné. Hotové aplikace v této kapitole budou zahrnovat základní funkcionality full-stack webových aplikací, jako například na front-endu přijetí dat, jejich zpracování, zobrazení a dále úprava dat uživatelem následovaná jejich posláním zpět na back-end. Poté si ukážeme zpracování dat v back-endové části aplikace, vystavení api rozhraní a principy komunikace a práce s databází.

5.1 Aplikace MERN

Jako nezbytnou podmínkou pro začátek tvorby této aplikace je nutné si nainstalovat NodeJS, pro tento projekt použijeme verzi v18.13.0, správce balíčků NPM a databáze MongoDB. Všechny tyto technologie nalezneme volně dostupné na internetu.

5.1.1 Založení nového projektu

Ze všeho nejdříve si vytvoříme novou složku projektu, ve které spustíme okno terminálu a zavoláme příkaz ***npx create-react-app app-name***, což je jeden z nejsnadnějších způsobů založení nové React aplikace. Tento nástroj byl vytvořen a je stále spravován vývojáři známé sociální sítě Facebook a slouží k zefektivnění a zrychlení tvorby nového projektu.

Terminál

```
npx create-react-app client // část příkazu app-name vyměníme za požadovaný název projektu
```

Dále si v kořenovém adresáři našeho projektu vytvoříme složku api, která bude sloužit jako back-endová část aplikace. Opět otevřeme terminál ve složce api a zavoláme příkaz ***npm init***, který vytvoří balíček npm, ve kterém budou následně ukládány další balíčky potřebné pro chod projektu. V terminálu zůstaneme a použijeme příkaz ***npm i <package-name>***, který nám stáhne jeden či více specifikovaných balíčků a nainstaluje je do projektu. Jako poslední příkaz zavoláme ***npm i -D nodemon***, ve kterém pomocí parametru

-D specifikujeme, že balíček nodemon bude pouze development-dependency, což znamená, že tento balíček bude využíván pouze při vývoji aplikace a po zbuildění (sestavení hotového projektu) bude vynechán, protože v produkčním prostředí nebude potřeba.

Terminál

```
npm init

npm i express mongoose cors // stáhne 3 balíčky najednou

npm i -D nodemon // stažení balíčku pouze pro vývojové prostředí
```

V poslední řadě si vytvoříme složku db, do které se budou ukládat dokumenty databáze MongoDB.

5.1.2 Technologie

Mongoose

Mongoose je objektově orientovaná javascriptová knihovna zprostředkovávající spojení mezi databází MongoDB a běhovým javascriptovým prostředím NodeJS. Tato knihovna poskytuje přímočaré řešení pro práci s daty aplikace, obsahuje vestavěnou validaci dat, tvorbu dotazů a další logiku pro usnadnění práce s MongoDB (Mongoose, 2023).

CORS

Cross-origin resource sharing je mechanismus umožňující příjem dat v jedné webové aplikaci z jiné domény, která tato data poskytnula (CORS, 2023). Cors je stejnojmenným balíčkem pro NodeJS, pomocí kterého lze spravovat tato povolení a je využíván například společně s technologiemi Express nebo Connect (cors, 2019).

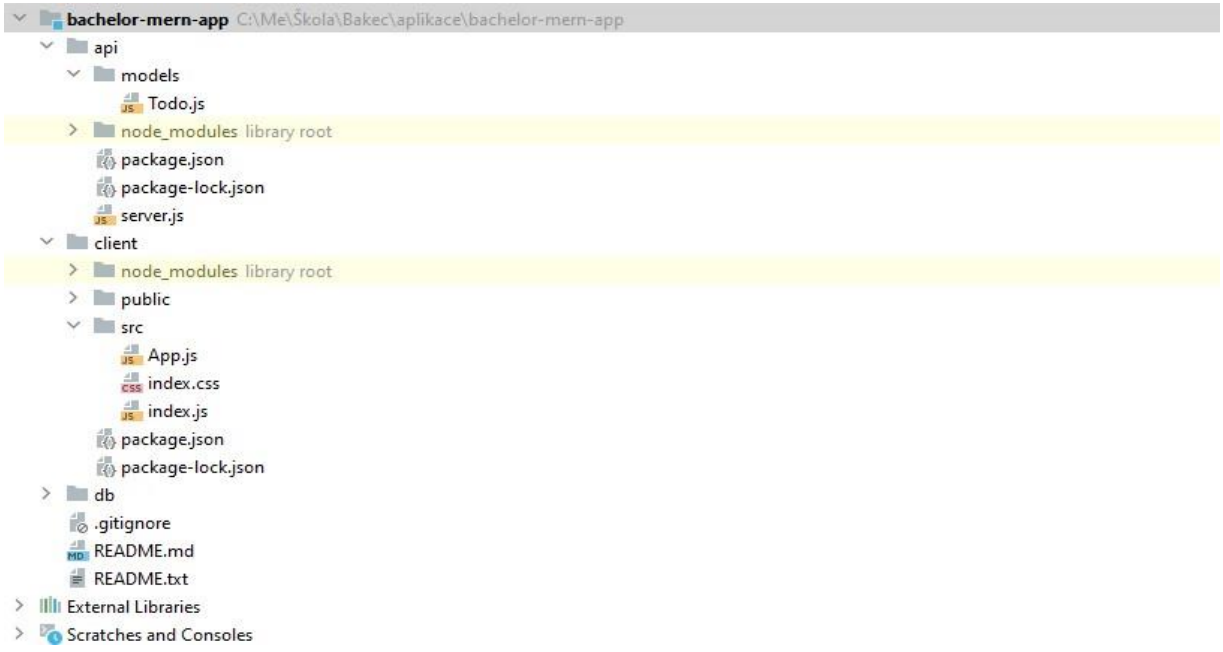
Nodemon

Nodemon je nástroj, který vývojářům usnadňuje vývoj webových aplikací založených na NodeJS automatickým restartováním aplikace ihned po zjištění změn souborů v projektu. Při vývoji tedy nemusíme po každé malé změně restartovat projekt ručně, což rapidně zrychluje a zpříjemňuje tvorbu aplikace (Nodemon, 2023).

5.1.3 Struktura aplikace

Vzhledem k tomu, že veškerou logiku píšeme ve frameworcích založených na JavaScriptu, strukturu celé aplikace sdružíme do jednoho projektu. Tento princip nám přináší výhodu jednotnosti a přispívá pohodlnějšímu a rychlejšímu vývoji celé aplikace.

Projekt logicky rozdělíme do adresářů podle vrstev aplikace. Podle obrázku níže můžeme vidět rozdělení struktury aplikace *bachelor-mern-app*, kterou jsme vytvořili pomocí návodu výše a následně upravili a doplnili o soubory pro naší potřebu podle nepsaných vývojářských konvencí. První vrstva s názvem *api* obsahuje serverovou část aplikace. Další část pojmenovaná *client* zahrnuje klientskou část viditelnou uživatelem v prohlížeči. Poslední část je složka *db* obsahující dokumenty databáze.



Obr. 1: Struktura MERN aplikace (vlastní zdroj)

1. **models**

   Adresář obsahující modely, v našem případě jediný *Todo.js*, které slouží jako obaly pro Mongoose schéma definující vlastnosti dokumentů, typy dat, defaultní hodnoty a validátory dat. Modely typu Mongoose poskytují rozhraní pro databázi k dotazování, vytváření, úpravu a mazání záznamů.

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const TodoSchema = new Schema({
    text:{
        type: String,
        required: true,
    },
    complete:{
        type: Boolean,
        default: false
    },
    timestamp:{
        type: String,
        default: Date.now()
    },
})

const Todo = mongoose.model("Todo", TodoSchema)

module.exports = Todo;
```

2. **api/node_modules**

   Adresář *node_modules* se objeví v projektu po spuštění příkazu ***npm install***, kterým nainstalujeme závislosti definované v souboru *package.json*. V tomto adresáři jsou uloženy všechny závislosti důležité pro chod aplikace. Pokud v našem projektu používáme verzovací nástroj git, poté by tento adresář měl být uveden v souboru *.gitignore*.

3. **api/package.json**

   Soubor ve formátu JSON obsahující všechny závislosti projektu a spustitelné skripty setříděné do přehledného seznamu. Obsahuje důležitý skript pro spuštění celého projektu. Vzhledem k definici závislostí uvnitř souboru můžeme projekt pomocí verzovacího nástroje sdílet s ostatními vývojáři bez nutnosti komitování

paměťově náročného adresáře *node_modules a* pomocí příkazu *npm install* nám dovoluje stáhnout všechny závislosti na jiném počítači.

4. **api/package-lock.json**

   Společně s *package.json* je tento soubor klíčovým prvkem definice závislostí projektu. Jedná se o soubor uchovávající informace o nainstalovaných balíčcích, ale oproti *package.json* je doplněn o dodatečné informace jako je například adresa, odkud se verze balíčku naposledy stáhnula.

5. **server.js**

   Jedná se o soubor, ve kterém je definována komunikace back-endu s databází a vystavení api rozhraní pro front-endovou část aplikace. Na ukázce kódu níže připravíme jednoduché připojení k databázi a následně vystavíme 4 různé funkce pro práci se záznamy v databázi.

   app.get()

   Metoda GET odkazuje na HTTP metodu, která je používána k vyžádání informací z určitého zdroje. V našem případě najde všechny objekty Todo v databázi a vrátí kompletní pole těchto objektů ve formátu JSON.

   app.post()

   Metoda POST je HTTP metoda, která je používána k posílaní dat na server. Tato metoda dovoluje poslání dat jako balíček v oddělené komunikaci se zpracovávajícím skriptem. Vzhledem k této separující logice POST metoda tedy zařídí, aby data nebyla vidět v URL. V ukázce níže tedy touto funkcí přijmeme data poslaná front-endem a po úspěšném zvalidování přidáme do databáze další objekt Todo.

app.delete()

Metoda DELETE je HTTP metoda, která vymaže z databáze záznam specifikovaný v parametru volání. V našem případě tedy nalezneme Todo objekt s identifikátorem shodným s identifikátorem přiloženým v parametru a vymažeme ho z databáze.

app.put()

Metoda PUT je HTTP metoda používaná k úpravě nějakého záznamu v databázi. Opět přijímá identifikátor, podle kterého najde vyžadovaný objekt a ten pak upraví. My pouze změníme boolean hodnotu na opačnou hodnotu.

Na posledním řádku specifikujeme, na jakém portu back-endová část aplikace naslouchá neboli na jakou adresu mají být posílány volání z front-endové části aplikace.

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");

const app = express();

app.use(express.json());
app.use(cors());

mongoose.connect("mongodb://127.0.0.1:27017/bachelor-mern-db", {
    useNewUrlParser: true,
    useUnifiedTopology: true
})
    .then(() => console.log("connected to db"))
    .catch(console.error);

const Todo = require("./models/Todo");

app.get("/todos", async (req,res) => {
    const todos = await Todo.find();

    res.json(todos)
})

app.post("/todo/new", (req,res) => {
    const todo = new Todo({
        text:req.body.text
    })

    todo.save();

    res.json(todo);
})

app.delete("/todo/delete/:id", async (req, res) => {
    const result = await Todo.findByIdAndDelete(req.params.id);

    res.json(result);
})

app.put("/todo/complete/:id", async (req,res) => {
    const todo = await Todo.findById(req.params.id);

    todo.complete = !todo.complete;

    todo.save();

    res.json(todo);

})

app.listen(3001, () => console.log("Server started on port 3001"))
```

6. **client/node_modules, client/package.json, client/package-lock.json**
   Tyto adresáře a soubory slouží ke stejné logice jako stejnojmenné soubory v adresáři api s tím rozdílem, že se starají o klientskou část aplikace na rozdíl od serverové části.

7. **public**
   Adresář obsahující obrázky použité v projektu a index.html soubor.

8. **src**
   Src je adresářem zahrnujícím všechen kód javascriptu. V komplexnějších aplikacích ho obvykle dále členíme na podadresáře vymezující stejnou logiku.

   App.js
   Tato komponenta je zobrazována funkcí *render()* poskytnutou knihovnou *react-dom* přímo do souboru HTML. Soubor *App.js* slučuje všechny další prvky aplikace, nejčastěji za pomoci routeru, který nebyl v rámci praktické části této práce implementován z důvodu redundance pro chod vybrané aplikace. Na příkladu níže můžeme vidět strukturu souboru *App.js*.

   V horní části ukázky jsou použity funkce **useState**, což jsou tzv. *hooky* umožňující spravovat stav ve funkcionální komponentě. Funkci useState můžeme předat jediný argument a to výchozí stav. UseState nám následně vrátí hodnotu aktuálního stavu a funkci pro aktualizaci tohoto stavu.

   Dále jsou implementovány funkce volající na rozhraní API, které si ukážeme v naší back-endové části. Získaná data následně používají k úpravě stavů hodnot v definovaných useState funkcích nebo naopak posílají data na server a upravují tak dokumenty v databázi.

V dolní části ukázky je použit **useEffect** hook. Tímto hookem říkáme funkcionální komponentě, aby reagovala na změny životního cyklu komponenty. V poli na konci funkce specifikujeme při změně, kterých parametrů se hook bude volat. V našem případě je pole prázdné, tudíž se hook zavolá jen jednou při zavedení komponenty do DOMu, provede funkci *getTodos()*, která natáhne data ze serveru a uloží je do useState hooku.

```
import {useState, useEffect} from "react";

const API_BASE = "http://localhost:3001";

function App() {
    const [todos, setTodos] = useState([]);
    const [popupActive, setPopupActive] = useState(false);
    const [newTodo, setNewTodo] = useState("");

    const getTodos = () => {
        fetch(API_BASE + "/todos")
            .then(res => res.json())
            .then(data => setTodos(data))
            .catch(err => console.error("Error: ", err))
    }
    const completeTodo = async id => {
        const data = await fetch(API_BASE + "/todo/complete/" + id, {
            method: "PUT"
        })
            .then(res => res.json())

        setTodos(todos => todos.map(todo => {
            if (todo._id === data._id) {
                todo.complete = data.complete;
            }

            return todo;
        }))
    }
    const deleteTodo = async id => {
        const data = await fetch(API_BASE + "/todo/delete/" + id, {
            method: "DELETE"
        }).then(res => res.json())

        setTodos(todos => todos.filter(todo => todo._id !== data._id))
    }

    const addTodo = async () => {
        const data = await fetch(API_BASE + "/todo/new/", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({
                text: newTodo.trim()
            })
        }).then(res => res.json())

        setTodos([...todos,data])
        setPopupActive(false)
        setNewTodo("")
    }

    useEffect(() => {
         getTodos();
     }, [])
```

Na následujícím kódu si ukážeme, jak je zapisován obsah uvnitř komponent Reactu. V návratové funkci komponenty používáme JSX, což je syntaktické rozšíření JavaScriptu umožňující psát značky podobné HTML uvnitř JavaScriptového souboru. Většina vývojářů preferuje stručnost této syntaxe, i přestože existují i alternativní způsoby zápisu komponent.

```
return (
    <div className="App">
        <h1>Welcome, Vojta</h1>
        <h4>Your tasks</h4>
        <div className="todos">
            {todos.map(todo => (
                <div className={"todo " + (todo.complete ? "is-complete" :
"")}
                     key={todo._id}
                     onClick={() => completeTodo(todo._id)}
                >
                    <div className="checkbox"></div>

                    <div className="text">{todo.text}</div>

                    <div className="delete-todo" onClick={(e) => {
                        e.stopPropagation();
                        deleteTodo(todo._id);
                    }}>x</div>
                </div>
            ))}
        </div>
        <div className="addPopup" onClick={() =>
setPopupActive(true)}>+</div>
        {popupActive && (
            <div className="popup">
                <div className="closePopup" onClick={() => {
                    setPopupActive(false)
                    setNewTodo("")
                }}>x</div>
                <div className="content">
                    <h3>Add Task</h3>
                    <input type="text"
                           className="add-todo-input"
                           onChange={(e) => setNewTodo(e.target.value)}
                           value={newTodo}/>
                </div>
                <button className={`button ${newTodo.trim() === "" &&
"disabled"}`}
                        disabled={newTodo.trim() === ""}
onClick={addTodo}>Create Task</button>
            </div>
        )}
    </div>
);
```

index.css

Soubor definující vizuální stylování prvků viditelných v prohlížeči.

index.js

Tělo adresářové struktury klientské části projektu obsahuje pouze jeden soubor nazvaný *index.js*, který je jako jediný spouštěn. Pro správný chod v něm musí být obsažena metoda z knihovny *react-dom* s názvem *render()*, která zajišťuje vykreslení obsahu do HTML souboru aplikace. Obvykle se zde renderuje pouze komponenta *App.js* obsahující všechny další prvky aplikace.

9. **.gitignore**

   Soubor definující názvy všech adresářů a souborů, které nemají být komitovány. Pokud v projektu nepoužíváme verzovací nástroj git, tento soubor do něho neobsadíme.

10. **README**

    Soubory obsahující důležité informace pro správné spuštění projektu.

5.2 Aplikace VueJS, Spring Boot a PostgreSQL

Stejně jako u předchozí aplikace musíme před začátkem tvorby nainstalovat NodeJS pro front-endovou část aplikace. Na rozdíl od aplikace MERN budeme pro naši back-endovou část potřebovat Java Development Kit neboli JDK, pro tento projekt konkrétně použijeme verzi 19.0.2. V poslední řadě si stáhneme databázový server PostgreSQL. Všechny použité technologie opět nalezneme volně dostupné na internetu.

5.2.1 Založení nového projektu

Front-endová část

V první řadě budeme opět potřebovat složku pro náš projekt, kde v okně terminálu zavoláme příkaz ***npm create vite@latest . -- --template template-name***. Vite je nástroj pro sestavování, který překlenuje propast mezi současným vývojem webu a vývojem webu nové generace a poskytuje rychlejší a výkonnější prostředí pro vývojáře a moderní webové projekty. V jádře dělá Vite v podstatě totéž, co *create-react-app*, hlavní rozdíl mezi těmito nástroji spočívá v tom, jak je kód servírován při vývoji a které moduly jsou podporovány. Na rozdíl od *create-react-app* si musíme po vytvoření projektu nástrojem Vite ještě nainstalovat závislosti příkazem ***npm install***.

Terminál

```
npm create vite@latest . -- --template vue // template-name nahradíme za vybraný framework

//dále se nás terminál zeptá na jméno projektu

? Package name: >> todo

npm install // nainstaluje závislosti definované v package.json
```

Back-endová část

Pro založení Spring Boot projektu použijeme Spring Initializr. Tento nástroj najdeme volně dostupný na oficiálních stránkach Springu a umožňuje nám si předvolit všechny důležité součásti a vlastnosti projektu a následně stáhnout soubor obsahující základní strukturu aplikace.

Typ projektu zvolíme Maven, jako programovací jazyk vybereme Javu, zaškrtneme nejnovější verzi Spring Bootu a verzi Javy podle našeho předem staženého JDK. Do závislostí projektu přidáme Spring Web, Spring Data JPA a PostgreSQL Driver.

5.2.2 Technologie

Maven

Maven je populární volně dostupný nástroj vyvinutý skupinou Apache Group, který umožňuje sestavovat, publikovat a nasazovat několik projektů najednou pro lepší správu projektů. Tento nástroj nám umožňuje sestavovat a dokumentovat rámec životního cyklu.

Spring Web

Balíček pro snadnější vytváření webových aplikací pomocí Spring MVC. Implementuje všechny základní funkce jádra Spring frameworku.

Spring Data JPA

Spring Data JPA usnadňuje vývoj aplikací, které používají Jakarta Persistence API (JPA) jako úložiště dat. Poskytuje podporu pro JPA a umožňuje snadný přístup ke zdrojům dat.

PostgreSQL Driver

Ovladač PostgreSQL umožňuje programům v jazyce Java připojit se k databázi PostgreSQL pomocí standardního, na databázi nezávislého kódu v jazyce Java.

5.2.3 Struktura aplikace

Vzhledem k tomu, že front-endovou část aplikace píšeme v JavaScriptovém frameworku a na back-endovou část používáme framework Javy, strukturu celé aplikace rozdělíme do 2 různých projektů. Rozdělením tímto způsobem oddělíme logiku klienta a serveru do separátních projektů.

Na následujícím obrázku si můžeme prohlédnout základní strukturu front-endové části aplikace, kterou jsme vytvořili po následování výše zmíněného návodu a následných malých úpravách pro správnou funkcionalitu naší aplikace.

Front-endová struktura

Obr. 2: Struktura Vue aplikace (vlastní zdroj)

1. **public**

   Adresář obsahující soubory mimo funkční logiku jako například obrázky. Oproti aplikaci front-endové části MERN aplikace, která je psaná v Reactu, adresář neobsahuje soubor index.html, který je uložen v adresáři projektu.

2. **src**

   Adresář src zastupuje obdobnou funkci jako stejnojmenný adresář u Reactu. Obsahuje veškerý kód javascriptu a v komplexnějších aplikacích je dále členěn na logické podadresáře.

   App.vue

   Tato komponenta je obdobou *App.js* v Reactu a zastupuje roli vstupního souboru aplikace. Vstupní bod JavaScriptu *main.js* importuje komponentu nejvyšší úrovně aplikace *App.vue*, takže do vstupního souboru JavaScriptu můžeme importovat jakékoli zásuvné moduly nebo integrace, aniž bychom zasáhli do logiky aplikace. Na přiloženém kódu níže můžeme nahlédnout do obsahu souboru *App.vue*.

   V horní části ukázky definujeme konstanty pomocí rozhraní **ref()**, které slouží ke tvorbě jednoduchých reaktivních hodnot. *Ref(initialValue)* voláme jako běžnou funkci přijímající jeden nepovinný argument jako výchozí hodnotu a ta nám vrací speciální objekt, ke kterému přistupujeme pomocí speciálního atributu *value*

(například todos.value). Po změně této hodnoty Vue znovu vykreslí komponentu, kde je hodnota zobrazena, aby odrážela novou aktuální hodnotu na obrazovce.

Dále můžeme pozorovat funkce volající na rozhraní API vystavené naší back-endovou částí. Oproti obdobným metodám v React části MERN aplikace se nijak zvlášť neliší a zastupují téměř stejnou funkcionalitu.

Jednou z mnoha funkcí dostupných ve Vue je funkce *watch*, která nám dovoluje sledovat stav aplikace a na základě těchto změn spouštět různé akce.

V poslední řadě v dolní části ukázky vidíme funkci *onMounted*, která je opět obsažena ve Vue a slouží k provedení nějakých akcí ihned při zavedení komponenty do DOMu. Funkce *onMounted* společně s funkcí *watch* jsou v Reactu nahrazeny funkcí *useEffect*.

```
<script setup>
import {ref, onMounted, computed, watch} from "vue";

const todos = ref([])
const isDisabled = ref(true)
const input_text = ref("")

const todos_asc = computed(() => todos.value.sort((a, b) => {
  let dateA = new Date(a.timestamp);
  let dateB = new Date(b.timestamp);
  return dateA - dateB
}))

const getTodos = () => {
  fetch("http://localhost:8080/api/todo")
      .then(res => res.json())
      .then(data => todos.value = data)
}

const tickTodo = (todo) => {

fetch(`http://localhost:8080/api/todo/${todo.id}?text=${todo.text}&complet
e=${todo.complete}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json;charset=utf-8'
    },
  })
      .catch(e => console.error(e))
}
const deleteTodo = (todoId) => {
  fetch(`http://localhost:8080/api/todo/${todoId}`, {
    method: 'DELETE',
  })
      .then(getTodos)
      .catch(e => console.error(e))
}

const addTodo = () => {
  if (input_text.value.trim() === "") {
    return
  }

  fetch('http://localhost:8080/api/todo', {
    method: 'POST',
    body: JSON.stringify({"text": input_text.value.trim(), "complete":
false}),
    headers: {
      'Content-Type': 'application/json;charset=utf-8'
    },
  })
      .then(getTodos)
      .catch(e => console.error(e))

  input_text.value = ""
}

watch(todos, () => {

}, {deep: true})

watch(input_text, () => {
  isDisabled.value = input_text.value.trim() === "";
}, {deep: true})

onMounted(() => {
  getTodos()
})

</script>
```

Klíčové rozdíly od Reactu přicházejí až při pohledu na syntaxi. Syntaxe frameworku Vue je ovlivněna jazykem AngularJS a zahrnuje běžné operace, jako jsou například smyčky for (v-for), podmíněné operace (v-else-if) a zpracování událostí (v-on). Bližší pohled nám poskytne ukázka kódu níže.

```
<template>
  <main class="app">
    <section class="greeting">
      <h2 class="title">VUE + SPRING + POSTGRES APP</h2>
    </section>
    <section class="create-todo">
      <h3>CREATE A TODO</h3>
      <form @submit.prevent="addTodo">
        <h4>What is on your todo list?</h4>
        <input
            type="text"
            placeholder="e.g. create a full stack app"
            v-model="input_text">
        <button :class="`addTodoInput ${isDisabled && 'disabled'}`"
type="submit" value="Add todo"
               :disabled='isDisabled'>Add note</button>
      </form>
    </section>
    <section class="todo-list">
      <h3>TODO LIST</h3>
      <div class="list">
        <div v-if="todos.length !== 0" v-for="todo in todos_asc"
:class="`todo-item ${todo.complete && 'done'}`">
          <label>
            <input type="checkbox" v-model="todo.complete"
@change="tickTodo(todo)"/>
            <span :class="`bubble ${todo.complete && 'complete'}`"></span>
          </label>
          <div class="todo-content"><p>{{ todo.text }}</p></div>
          <div class="actions"><button class="delete"
@click="deleteTodo(todo.id)">Delete</button></div>
        </div>
        <div v-else>There are no notes left, everything is done.</div>
      </div>
    </section>
  </main>
</template>
```

main.css

Soubor definující vizuální stylování prvků viditelných v prohlížeči.

main.js

Vstupní bod HTML načítá aplikaci z tohoto souboru (obdobně jako v Reactu index.js). Každá aplikace Vue začíná vytvořením nové instance aplikace pomocí funkce *createApp*. Instance aplikace nebude nic vykreslovat, dokud nebude zavolána její metoda *mount()*, která připojí instanci aplikace do kontejnerového prvku.

3. **vite.config.js**

   Vite podporuje více front-endových frameworků. Z tohoto důvodu je při založení projektu v kořenovém adresáři vytvořen konfigurační soubor vite.config.js, který specifikuje, že vyvíjíme konkrétně projekt Vue.

Back-endová struktura

Obr. 3: Struktura Spring Boot aplikace (vlastní zdroj)

1. **.idea**

   Tento adresář je vytvořen po otevření projektu ve vývojovém prostředí IntelliJ IDEA a obsahuje konfigurační soubory, které se mohou u každého vývojáře lišit, proto ho nekomitujeme pomocí gitovacího nástroje.

2. **.mvn**

   Adresář obsahující soubory, které pomáhají Mavenu pochopit, jaký typ balíčku má být v projektu použit. U projektů, které potřebují konkrétní verzi Mavenu, nemusíme Maven instalovat lokálně, ale místo instalace mnoha verzí lze použít pouze obalový skript specifický pro projekt, který je k dispozici při vytváření projektu Spring Boot.

3. **src**

   Stejně jako je tomu u každého jiného projektu, nejdůležitější složkou je src. Obsahuje všechen zdrojový kód a zdroje tvořící aplikaci. Nalezneme zde hlavní spustitelný soubor aplikace.

TodoApplication

Spustitelná třída obsahující funkci *main()*, která slouží jako vstupní bod samotné Java aplikace.

resources

Obsahuje všechny soubory s prostředky, které jsou potřebné ke spuštění aplikace.

application.properties

Konfigurační soubor aplikace, který slouží k úpravě výchozích konfigurací systému. Jedná se o způsob, jak přizpůsobit aplikaci Spring, a nemá nic specificky společného se Spring Bootem. V našem případě tento soubor využijeme k připojení k databázi PostgreSQL a ke specifikaci různých vlastností připojení.

Obr. 4: Soubor application.properties (vlastní zdroj)

todo

Balíček, který logicky sdružuje všechny třídy a rozhraní starající se o funkcionalitu spojenou s Todo.

a. **Todo**

   Todo je třída, která definuje data a metody. Aby JPA balíček věděl o této entitě, musíme na úrovni třídy specifikovat anotaci *@Entity*. Musíme také zajistit, aby entita měla konstruktor bez příznaků a primární klíč. Ve většině případů nebude název tabulky v databázi a název entity stejný. V těchto případech můžeme název tabulky zadat pomocí anotace *@Table*

(@Table(name="TODO")). Každá entita JPA musí mít primární klíč, který ji jednoznačně identifikuje. Primární klíč definuje anotace *@Id* a anotace *@SequenceGenerator* definuje generátor primárního klíče, na který se lze odkazovat podle názvu, když je pro anotaci *@GeneratedValue* zadán prvek generátoru. Identifikátory můžeme generovat různými způsoby, které určuje anotace *@GeneratedValue*.

```java
@Entity
@Table
public class Todo {
    @Id
    @SequenceGenerator(
            name = "todo_sequence",
            sequenceName = "todo_sequence",
            allocationSize = 1
    )
    @GeneratedValue(
            strategy = GenerationType.SEQUENCE,
            generator = "todo_sequence"
    )
    private Long id;
    private String text;
    private Boolean complete;
    private LocalDateTime timestamp;
    public Todo() {
        this.timestamp = LocalDateTime.now();
    }
    public Todo(String text, Boolean complete) {
        this.text = text;
        this.complete = complete;
        this.timestamp = LocalDateTime.now();
    }
    public Todo(Long id, String text, Boolean complete) {
        this.id = id;
        this.text = text;
        this.complete = complete;
        this.timestamp = LocalDateTime.now();
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
    public Boolean getComplete() {
        return complete;
    }
    public void setComplete(Boolean complete) {
        this.complete = complete;
    }
    public LocalDateTime getTimestamp() {
        return timestamp;
    }
    public void setTimestamp(LocalDateTime timestamp) {
        this.timestamp = timestamp;
    }
```

b. **TodoConfig**

Třída vytvořená za účelem testování při vývoji, která po spuštění aplikace přidá do databáze 2 specifikované objekty.

Anotaci *@Bean* použijeme na metodu, aby určila, že vrací bean, který má být spravován kontextem Spring. Bean je objekt, který je instancován a sestaven po spuštění projektu. Tuto anotaci obvykle deklarujeme v metodách konfiguračních tříd. V takovém případě mohou metody beanu odkazovat na jiné metody *@Bean* ve stejné třídě jejich přímým voláním.

```
@Configuration
public class TodoConfig {

    @Bean
    CommandLineRunner commandLineRunner(TodoRepository repository)
{
        return args -> {
            Todo todo1 = new Todo("Clean the bathroom",true);
            Todo todo2 = new Todo("Cut the grass",false);
            repository.saveAll(List.of(todo1,todo2));
        };
    }
}
```

c. **TodoController**

Tato třída je zodpovědná za vystavení REST API rozhraní, ve kterém specifikujeme, na jaké adresy se front-end dotazuje pro získávání a posílání dat. Vypisujeme zde jednotlivé metody, které udávají, zda jsou k volání potřeba parametry a jaké další metody se následně zavolají. Spring 4.0 zavedl anotaci *@RestController*, aby zjednodušil vytváření webových služeb RESTful. Jedná se o praktickou anotaci, která kombinuje *@Controller* a *@ResponseBody*, díky čemuž odpadá nutnost anotovat každou metodu obsluhy požadavku ve třídě controlleru anotací *@ResponseBody*.

Anotace *@CrossOrigin* nám umožňuje sdílení prostředků mezi různými původy. Ve výchozím nastavení povoluje všechny původy, všechny hlavičky a metody HTTP uvedené v anotaci *@RequestMapping*. Jedná se o alternativu

ke cors balíčku použitém v naší MERN aplikaci. Anotace *@Autowired* používáme pro automatické vkládání závislostí.

```
@CrossOrigin
@RestController
@RequestMapping(path = "api/todo")
public class TodoController {

    private final TodoService todoService;

    @Autowired
    public TodoController(TodoService todoService) {
        this.todoService = todoService;
    }

    @GetMapping
    public List<Todo> getTodos(){
        return todoService.getTodos();
    }

    @PostMapping
    public void addNewTodo(@RequestBody Todo todo) {
        todoService.addNewTodo(todo);
    }

    @DeleteMapping(path = "{todoId}")
    public void deleteTodo(@PathVariable("todoId") Long todoId){
        todoService.deleteTodo(todoId);
    }

    @PutMapping(path = "{todoId}")
    public void updateTodo(
            @PathVariable("todoId") Long todoId,
            @RequestParam(required = false)String text,
            @RequestParam(required = false)Boolean complete){
        todoService.updateTodo(todoId,text,complete);
    }
}
```

d. **TodoRepository**

Rozhraní, které se nám stará o přístup k databázi. Repository je jmenná konvence pro pojmenování těchto rozhraní pracujících s databází. Anotaci *@Repository* využíváme k označení třídy, která poskytuje mechanismus pro ukládání, načítání, vyhledávání, aktualizaci a mazání objektů.

```
@Repository
public interface TodoRepository extends JpaRepository<Todo,Long> {}
```

e. **TodoService**

V této třídě nalezneme metody, které jsou volány ze třídy TodoController. Tyto metody obsahují stěžejní logiku, která specifikuje, co se má po přijetí volání stát, zda se data mají v databázi nalézt, vytvořit, upravit či smazat. Pomocí anotace *@Transactional* můžeme metodu zabalit do databázové transakce. To nám umožní nastavit podmínky propagace, izolace, časového limitu, čtení a návratu pro naši transakci.

```
@Service
public class TodoService {

    private final TodoRepository todoRepository;

    @Autowired
    public TodoService(TodoRepository todoRepository) {
        this.todoRepository = todoRepository;
    }

    public List<Todo> getTodos(){
        return todoRepository.findAll();
    }

    public void addNewTodo(Todo todo) {
        todoRepository.save(todo);
    }

    public void deleteTodo(Long todoId) {
        boolean exists = todoRepository.existsById(todoId);
        if (!exists) {
            throw new IllegalStateException("Todo with id " +
todoId + " does not exists");
        }
        todoRepository.deleteById(todoId);
    }

    @Transactional //entity goes to manage state
    public void updateTodo(Long todoId, String text, Boolean
completed) {
        Todo todo = todoRepository.findById(todoId).orElseThrow(()
-> new IllegalStateException(
                "Todo with id " + todoId + " does not exist."
        ));
        if (text != null && text.length() > 0) {
            todo.setText(text);
        }
        if (completed != null && todo.getComplete() != completed) {
            todo.setComplete(completed);
        }
    }
}
```

4. **target**

   Target je výchozí výstupní složka Mavenu. Po sestavení nebo zabalení projektu se do této složky vygeneruje finální zabalený soubor s příponou .jar nebo .war, který můžeme použít k publikování webové aplikace.

5. **pom.xml**

   Pom je soubor typu xml, který obsahuje informace o projektu a konfigurační údaje používané nástrojem maven k sestavení projektu. Soubor POM (Project Object Model) je základní jednotkou práce v Mavenu. Obsahuje informace jako groupId a artifactId projektu, verzi projektu, závislosti, zásuvné moduly a informace o zdrojích.

6 Shrnutí výsledků

V teoretické části práce jsme čtenáře nejprve obeznámili s historií webových aplikací, kterou jsme následovali základní strukturou aplikace s popsáním jednotlivých částí front-endu, back-endu a databáze. Následně jsme vysvětlili pojem framework a seznámili čtenáře s jeho zástupci. V praktické části práce jsme využili poznatky získané z teoretické části a následně vytvořili dvě aplikace založené na různých technologiích. V této kapitole zhodnotíme použité technologie a porovnáme hotové aplikace.

Architektura MERN je populární a vysoce používaná kombinace technologií pro vývoj moderní webové aplikace. Mezi její výhody patří především skutečnost, že všechny čtyři technologie jsou psané v JavaScriptu. Takto postavená aplikace dovoluje lepší sdílení kódu mezi front-endem a back-endem, usnadňuje správu aplikace, zkracuje celkovou dobu vývoje a umožňuje vývojáři se soustředit na jeden programovací jazyk a maximalizovat tak potenciál tvorby takového projektu. V architektuře MERN jsme na front-endové části použili knihovnu React, což je velice populární javascriptová knihovna, kterou můžeme využít jak v jednodušších projektech, tak v robustních a komplexních aplikacích. Jednu z hlavních výhod, která plyne z vysoké popularity, představuje obrovská komunita vývojářů, která vysoce usnadňuje křivku učení. Tato výhoda nám přináší i jednu velkou nevýhodu, a to neustálenou metodologii. React lze psát mnoha způsoby, a proto je občas obtížné sesynchronizovat tým vývojářů podílejících se na vývoji jednoho projektu nebo přechod jednoho vývojáře mezi projekty. Mezi výhody Reactu patří intuitivní syntaxe JSX, komponentová architektura a správa stavů na úrovni komponent, avšak mnoho aplikací se spoléhá na knihovny třetích stran, které vytvořila a podporuje komunita, což přináší další nevýhodu, a to fakt, že výběr té správné knihovny může být pro začátečníky výzvou. Tato nevýhoda je u Vue eliminováná tím, že důležité nástroje byly vytvořeny a jsou stále podporovány hlavním vývojovým týmem. Back-endovou část aplikace MERN jsme vytvořili propojením technologií Express a Node. Node je populární a výkonná serverová platforma pro JavaScript. Express je často označován jako rychlý, neokoukaný a minimalistický webový framework, který běží uvnitř serveru Node a disponuje výkonnými modely pro směrovaní URL a dále zpracováním požadavků HTTP a řízením jejich odpovědí. Hlavní výhoda kombinace Express a Node oproti frameworkům Javy spočívá v rychlosti zpracovaných požadavků za určitou dobu. Počet požadavků za

sekundu je rapidně vyšší, což nám přináší rychlejší obsluhu stránek, protože Node spolu s Expressem dokáže zpracovat obrovské množství současných připojení a souběžných akcí. Naopak mezi nevýhody můžeme zařadit fakt, že Express je mnohem „lehčí“ framework než Spring Boot, a to má za následek nutnost psát mnohem více kódu. Je těžší si tak udržet spořádaný a čitelný projekt. Vývojáři pracující ve velkých týmech si musí definovat standardy, které je nutné dodržovat. V databázové části aplikace jsme použili dokumentovou databázi MongoDB, která se řadí mezi vysoce používané technologie. Umožňuje ukládat data v nestrukturované formě, konkrétně pomocí dokumentů JSON, poskytuje vysoký výkon a dokáže pracovat až stokrát rychleji než jiné relační databáze. Mezi výhodami musíme zmínit horizontální škálovatelnost, kterou můžeme zajistit přidáním více serverů pro ukládání dat, a tím zlepšíme výkon aplikace. Na druhou stranu dokumentové databáze nepodporují vztahy mezi datovými typy, a proto jsou pro náročné nebo složité transakce vhodnější SQL databáze, protože jsou stabilnější a zajišťují integritu dat.

Pro front-endovou část druhé aplikace jsme využili javascriptového frameworku Vue, který je používán pro vytváření uživatelských rozhraní a jednostránkových aplikací. Stejně jako React i Vue poskytuje otevřený zdrojový kód a jak jsme již zmínili u předchozí aplikace, narozdíl od Reactu jsou nezbytné funkce pokryty oficiálními knihovnami frameworku. Syntaxí založenou na HTML a JavaScriptu nám Vue přinesl intuitivní a snadno naučitelný způsob pro tvorbu komponent. Používá takzvané direktivy neboli speciální atributy HTML, které slouží například k propojení datového modelu s vlastnostmi DOM prvku. Ačkoliv komunita Vue neustále roste, její velikost je stále menší než u konkurenčního React, proto existuje méně knihoven a zdrojů třetích stran. Značné množství materiálů je napsáno v čínštině a stále potřebuje překlad do anglického či českého jazyka. Back-endovou vrstvu této aplikace jsme vytvořili za pomocí Spring Bootu, který jsme zvolili z důvodu porovnání Javy a JavaScriptu na serverové straně aplikace. Spring Boot je framework pro tvorbu webových aplikací v jazyce Java a jeho cíl spočívá ve zjednodušení procesu vývoje aplikací, snadného nasazení a konfigurace. Hlavní výhodou Javy je univerzálnost, zatímco Node byl navržen pro použití v prohlížeči. Java s sebou tedy nese obrovské knihovny kódu a můžeme ji použít k vývoji softwaru pro různé typy platforem, zatímco Node použijeme pouze k vytváření webových aplikací.

Nejvýznamnější nevýhodou Spring Bootu je, že na rozdíl od Node JS spotřebovává velké množství paměti. V důsledku toho vznikají vyšší nároky na paměť zařízení. Mezi další nevýhody bychom měli určitě zařadit složitější debuggování než u první aplikace a s tím spojenou složitější křivku učení. Databázovou vrstvu této aplikace jsme vyplnili technologií PostgreSQL. Navzdory popularitě NoSQL databází jsou relační databáze stále důležité pro různé aplikace vzhledem ke své robustnosti a silným dotazovacím schopnostem. Relační databáze se skvěle hodí pro provádění složitých dotazů a reportování na základě dat v případech, kdy se struktura dat často nemění. Databáze s otevřeným zdrojovým kódem, jako je PostgreSQL, nám nabízejí cenově výhodnou alternativu jako stabilní databáze produkční třídy. Klíčovou vlastností, která MongoDB odlišuje od PostgreSQL, je přístup k ukládání dat. Protože MongoDB není relační, používá místo tabulek kolekce. PostgreSQL je kompatibilní s SQL, podporuje cizí klíče a povolením jejich omezení můžeme zabránit vkládání neplatných dat do sloupců cizích klíčů.

7 Závěry a doporučení

Po pečlivém zvážení výsledků zmíněných výše nemůžeme jednoznačně rozhodnout, zda některá z architektur je ve všech ohledech lepší než jiná, vzhledem k rozmanitosti potřeb webových aplikací. V podstatě bychom však mohli doporučit architekturu MERN samostatným vývojářům, kteří se chtějí zdokonalit v JavaScriptu a kteří chtějí mít pod kontrolou všechny vrstvy své aplikace, nebo menším týmům. Hotové projekty se nejvíce hodí pro aplikace pracující v reálném čase, které vyžadují vysokou rychlost odezvy. Architektura spojující technologie Vue, Spring Boot a PostgreSQL přináší výhodu pro větší firmy, které mají oddělené vývojářské týmy pro front-end a back-end. Z použití těchto technologií nejvíce benefitují rozsáhlé systémy vyžadující vysokou míru bezpečnosti a integrity dat, které využívají například banky nebo rozsáhlé skladové systémy.

8 Seznam použité literatury

1. SHKLAR, L. a R. ROSEN. *Web Application Architecture: Principles, Protocols and Practices* [online]. B.m.: Wiley, 2003. ISBN 978-0-471-48656-5.
2. HOLZNER, Steven. JavaScript profesionálně: [kompletní referenční příručka]. Praha: Mobil Media, c2003, iDnes internet knihy. ISBN 80-86593-40-1
3. MALÍK, Pavol. *Moderní architektury webových aplikací* [online]. Brno, 2019 [cit. 2023-01-26]. Dostupné z: https://theses.cz/id/di5jln/. Diplomová práce. Vysoké učení technické v Brně.
4. BROOKS, D.R. *Guide to HTML, JavaScript and PHP: For Scientists and Engineers* [online]. B.m.: Springer London, 2014. ISBN 978-1-4471-6060-1.
5. BROOKS, D.R. *An Introduction to HTML and JavaScript: for Scientists and Engineers* [online]. B.m.: Springer London, 2007. Computer science. ISBN 978-1-84628-656-8.
6. HEIDERICH, M. Web Application Obfuscation [online]. B.m.: Elsevier/Syngress, 2011. ISBN 978-1-282-95605-6.
7. Node JS [online]. Dostupné z URL: https://nodejs.org/
8. Express JS [online]. Dostupné z URL: https://expressjs.com/
9. MongoDB [online]. Dostupné z URL: https://www.mongodb.com/
10. React JS [online]. Dostupné z URL: https://reactjs.org/
11. Vue JS [online]. Dostupné z URL: https://vuejs.org/
12. Angular [online]. Dostupné z URL: https://angular.io
13. Spring [online]. Dostupné z URL: https://spring.io
14. JavaScript | MDN [online]. Dostupné z URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript
15. MASON, Philip. *SAS Stored Processes: A Practical Guide to Developing Web Applications*. B.m.: Apress, 2020. ISBN 978-1-4842-5925-2.

16. ROJAS, C. *Building Progressive Web Applications with Vue.js: Reliable, Fast, and Engaging Apps with Vue.js* [online]. B.m.: Apress, 2019. ISBN 978-1-4842-5334-2.
17. PELZER, D. *A Modular Framework for Optimizing Grid Integration of Mobile and Stationary Energy Storage in Smart Grids.* [online]. B.m.: Springer Vieweg Wiesbaden, 2019. ISBN 978-3-658-27024-7.
18. Mongoose [online]. Dostupné z URL: https://mongoosejs.com
19. CORS [online]. Dostupné z URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
20. cors [online]. Dostupné z URL: https://www.npmjs.com/package/cors
21. Nodemon [online]. Dostupné z URL: https://www.npmjs.com/package/nodemon
22. PostgreSQL [online]. Dostupné z URL: https://www.postgresql.org/about/
23. VOHRA, D. *Kubernetes Microservices with Docker.* [online]. B.m.: Apress, 2016. ISBN 978-1-4842-1906-5.

9 Seznam obrázků

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2021/2022

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Vojtěch Kycelt**
Osobní číslo:	**I2000383**
Adresa:	Zámecká 609, Chlumec nad Cidlinou - Chlumec nad Cidlinou IV, 50351 Chlumec nad Cidlinou, Česká republika
Téma práce:	Architektura webových aplikací
Téma práce anglicky:	Web application architecture
Jazyk práce:	Čeština
Vedoucí práce:	doc. Ing. Filip Malý, Ph.D. Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Cílem této bakalářské práce je získání přehledu o využívaných technologiích při tvorbě webových aplikací, definování vhodné architektury pro konkrétní případy a seznámení budoucích vývojářů i ostatních čtenářů s danou problematikou. V praktické části práce pak vyvinout 2 různé aplikace s rozdílnými technologiemi za použití poznatků z teoretické části práce, jejich porovnání a využití.

Osnova:

1. Úvod 2. Webová aplikace 2.1 Historie webových aplikací 3. Webový rámec 4. Struktura webové aplikace 4.1 Klientská část 4.2 Server 4.3 Databázová část 5. Praktická část 5.1 Aplikace A 5.2 Aplikace B 6. Shrnutí 7. Závěry a doporučení 8. Seznam použité literatury 9. Přílohy

Seznam doporučené literatury:

Podpis studenta: Kycelt Datum: 13.4.2023

Podpis vedoucího práce: Datum: 13.4.2023