VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÝ EDITOR PREZENTACÍ
PRESENTATIONS WEB EDITOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ADAM ABRAHÁM**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Abrahám Adam**
Program: Informační technologie
Název: **Webový editor prezentací**
Presentations Web Editor
Kategorie: Web

Zadání:

1. Prostudujte současné technologie pro tvorbu webových aplikací s tlustým klientem v JavaScriptu.
2. Seznamte se s existujícími nástroji a knihovnami pro tvorbu prezentací na webu v HTML a JavaScriptu, např. Reveal.js a další.
3. Po dohodě s vedoucím navrhněte architekturu webové aplikace pro interaktivní tvorbu prezentací s využitím existujících knihoven.
4. Implementujte navrženou aplikaci na vhodné platformě. Implementujte podporu exportu výsledné statické prezentace a verzování.
5. Proveďte testování vytvořené aplikace.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

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

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 23. října 2020

Abstrakt

Cieľom tejto práce je implementácia webovej aplikácie s tlstým klientom pre spravovanie prezentácií s obsahom typu Markdown, ktorý sa následne prezentuje pomocou prezentačného rámca(framework) Reveal.js. Frontend aplikácie je vytvorený pomocou Vue.js s nadstavbou Nuxt.js, backend pomocou Express.js a ako úložisko dát je zvolená NoSQL dokumentová databáza MongoDB. Frontend a backend časti aplikácie komunikujú medzi sebou cez technológiu REST. Výsledná aplikácia umožňuje užívateľom zobrazovať, upravovať a vytvárať viacero verzií danej prezentácie. Práca naďalej obsahuje popis, porovnanie súčasných technológií a zdôvodnenie ich výberu.

Abstract

The aim of this thesis is to implement a web application to manage presentations with Markdown content, which are then presented through slideshow framework Reveal.js. Frontend of the application is created with Vue.js and Nuxt.js, backend with Express.js and for data storage I have chosen MongoDB, a NoSQL document database. Frontend and backend parts of the application communicate with each other through REST technology. The application allows users to view, edit and create more versions of the same presentation. This thesis furthermore contains description, comparison of current technologies and substantiation of their selection.

Kľúčové slová

Webová aplikácia, Editor, Prezentácia, Markdown, JavaScript, TypeScript, NoSQL, MongoDB, Node.js, Express.js, REST, Vue.js, Nuxt.js, Buefy, Reveal.js, Cypress, Postman, Git, Github

Keywords

Web application, Editor, Presentation, Markdown, JavaScript, TypeScript, NoSQL, MongoDB, Node.js, Express.js, REST, Vue.js, Nuxt.js, Buefy, Reveal.js, Cypress, Postman, Git, Github

Citácia

ABRAHÁM, Adam. *Webový editor prezentací*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Webový editor prezentací

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana profesora Radeka Burgeta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Abrahám
9. mája 2021

Poďakovanie

Moje poďakovanie patrí vedúcemu práce, profesorovi Radekovi Burgetovi za jeho rady, doporučenia a za pomoc pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Táto bakalárska práca sa zaoberá implementáciou webovej aplikácie pre správu webových prezentácií. Hlavnou výhodou oproti konkurenčným riešeniam je možnosť kopírovania snímok medzi prezentáciami a verzovania jednotlivých prezentácií. Týmto spôsobom sa vyhneme zbytočnému ukladaniu našej práce pod názvy `xy-final`, `xy-final-final` a podobne, s ktorými sme sa už určite viacerí stretli. Aplikácia umožňuje uchovať všetky naše predošlé verzie práce, ktoré sú dostupné na jednom mieste. Cieľom aplikácie je umožniť užívateľom vytváranie a spravovanie prezentácií pomocou značkovacieho jazyka Markdown.

Práca sa skladá zo siedmich kapitol. Druhá kapitola(2) oboznámi čitateľa so súčasnými webovými technológiami. Popisuje ich účel a porovnáva ich vlastnosti.

Tretia kapitola(3) sa už zaoberá použitými technológiami v aplikácii. Na úvod sa opisuje nerelačná dokumentová databáza MongoDB(3.1.1) a pomocná knižnica pre modelovanie objektov Mongoose(3.1.2). Po popise zvolenej databáze nasleduje úvod do TypeScriptu(3.2), pri ktorom sa čitateľ zoznámi s jednotlivými typovými systémami a ich výhodami. Ďalej v kapitole sa uvádzajú použité frontendové technológie, výhody CSS preprocessora Sass(3.3.2) a CSS rámca Buefy(3.3.2). V kapitole je popísaná myšlienka za voľbou frontendového rámca Vue.js(3.3.3) s nadstavbou Nuxt.js(3.3.5) a technológiou Composition API(3.3.4). Pre serverovú časť sa použil Node.js(3.4.1) s rámcom Express.js(3.4.1). Aplikačné rozhranie sa realizovalo pomocou architektúry REST(3.4.2).

Štvrtá kapitola(4) popisuje celkovú štruktúru aplikácie(4.2). Nachádza sa v nej podrobný návod na použitie aplikácie(4.1), myšlienka za návrhom užívateľského rozhrania(4.3.2) a databáze(4.4). Čitateľ sa oboznámi s výhodami vzoru repozitára(4.5) a s tabuľkou koncových bodov aplikácie s ich popisom(4.1).

Piata kapitola(5) sa zaoberá konkrétnou implementáciou klientskej(5.2) a serverovej(5.3) časti aplikácie. Popisuje sa v nej organizácia úloh a verzovanie zdrojového kódu cez nástroj GitHub(5.1).

Dôležitou súčasťou implementácie bolo priebežné testovanie aplikácie, pre zaručenie očakávaného správania. Popis typov testovaní a použitých nástrojov sa nachádza v kapitole 6.

Práca je ukončená kapitolou 7, zhrnutím dosiahnutých výsledkov a nápadom na možné rozšírenie aplikácie do budúcna.

Kapitola 2

Popis a porovnanie súčasných webových technológií

Cieľom tejto kapitoly je informovať čitateľa o súčasných webových technológiách. Kapitola obsahuje krátky popis jednotlivých jazykov a technológií a ich porovnanie. Záver popisuje najpopulárnejšie pomocné webové rámce (ďalej označované ako `frameworky`) pre vytváranie prezentácií vo webovom prehliadači.

2.1 Databáza

Databáza je množina štrukturovaných údajov ktorá slúži na uloženie informácií[16]. Tieto informácie si môže počítačový program, alebo človek pomocou dopytovacieho jazyka jednoducho získať.

Aktuálnym najznámejším dopytovacím jazykom je `SQL`. Počítačový program, ktorý slúži na vytváranie dopytov sa nazýva `Systém riadenia bázy dát`. Štyri základné operácie nad záznamami databázy sa označujú skratkou `CRUD`, ktorá odpovedá anglickým pojmom v preklade vytvoriť, čítať, upravovať a mazať.

Rozlišujeme relačné a nerelačné databázy, ktoré sú podrobnejšie popísané nižšie.

2.1.1 Relačná databáza

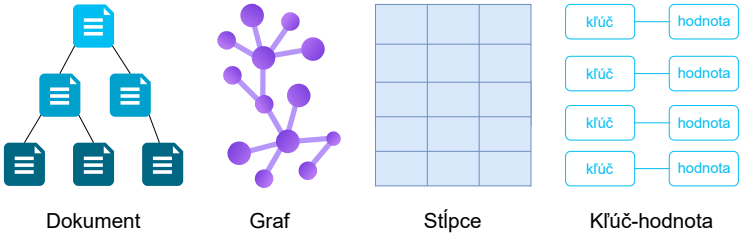
Relačná databáza je typ databázy, ktorá funguje na princípe tabuliek riadkov zoskupených do jednotlivých vzťahov. V relačnej databáze je každý riadok v tabuľke nazývaný ako záznam a každý záznam má pridelený unikátny kľúč `ID`. Stĺpce tabuľky obsahujú atribúty údajov.

2.1.2 Nerelačná databáza

Nerelačná databáza sa označuje aj ako `NoSQL`[1] databáza. Táto technológia je medzi nami už mnohé roky, ale v súčasnej dobe naberá prudký rast popularity kvôli meniacemu sa prostredia údajov. Vývojári sa potrebujú adaptovať, aby si vedeli poradiť so širokou škálou údajov a ich objemom. NoSQL databázy sú efektívne pri spracovaní veľkých objemov vzájomne nesúvisiacich, alebo rýchle sa meniacich údajov s mimoriadnou rýchlosťou dopytov[9]. Sú vhodné pre rýchly a flexibilný vývoj aplikácií. Nepoužívajú tabuľkovú schému riadkov

[1] https://azure.microsoft.com/cs-cz/overview/nosql-database/

a stĺpcov, ale model, ktorý je navrhnutý pre konkrétne požiadavky typu uložených údajov. Na vytvorenie dopytov sa nepoužíva jazyk `SQL`, ale iné programovacie jazyky. Údaje sú nezávislé na schéme, na rozdiel od relačných databáz, kde je schému potrebné udržiavať. Dáta môžu byť uložené ako páry kľúč-hodnota, ako JSON (*JavaScriptový objektový zápis*[2]) dokumenty, ako stĺpce alebo ako graf skladajúci sa z uzlov a hrán. Ich rozdiel je znázornený na obrázku 2.1. V tejto práci sa používa dokumentová databáza `MongoDB`, jej podrobnejší popis sa nachádza v sekcii 3.1.1.



Obr. 2.1: Ilustrácia typov NoSQL databáz [9].

2.2 Frontend

Frontend, alebo častokrát nazývaný aj ako klient, je prezentačná vrstva aplikácie ktorá poskytuje používateľské rozhranie. Je to časť, ktorú užívateľ vidí, s ktorou pracuje a ovláda. Medzi najpopulárnejšie technológie pre vývoj frontendu patrí HTML, CSS a JavaScript, pomocou ktorých je táto aplikácia implementovaná.

2.2.1 JavaScript

JavaScript je objektovo orientovaný skriptovací jazyk. Najviac sa používa pri vývoji dynamických webových aplikácií, ale využíva sa aj pri tvorbe multiplatformových stolových aplikácií a hybridných mobilných aplikácií. Poskytuje interaktivitu nad obsahom.

Medzi jeho výhody patrí dynamické načítanie stránky, zmena obsahu bez fyzického znova načítania stránky a jeho rozšíriteľnosť. JavaScript sa hlavne zameriaval na klientskú časť aplikácie, kým nenastúpil na scénu framework `Node.js`(3.4.1) a podobné serverové technológie, účelom ktorých je tvorba serverovej časti aplikácie.

Existuje niekoľko JavaScriptových frameworkov pre uľahčenie vývoja, pomocou ktorých sa vyhneme takzvaným biolerplate[3] kódom. Medzi najpopulárnejšie frameworky patrí Vue.js, React, Angular a Node.js.

2.2.2 Vue.js

Vue.js[4] je JavaScriptový framework, ktorý patrí medzi najpopulárnejšie nástroje pre tvorbu jednostránkových aplikácii[15]. Je výkonný, rýchly a minimalistický. Stal sa jedným z obľúbených kvôli jeho jednoduchosti. Má veľmi prehľadnú a pestrú dokumentáciu v anglickom aj čínskom jazyku.

[2] JavaScript Object Notation - spôsob zápisu údajov do objektov
[3] Kód ktorý sa musí opakovane vyskytovať na viacerých miestach bez významnej zmeny
[4] `https://vuejs.org/`

Framework bol vytvorený Evanom Youom, ktorý pracoval v Googli a Meteore. V Googli začal používať framework AngularJS, ktorý mu avšak nevyhovoval kvôli jeho striktnému písaniu kódu. Jeho motiváciou bolo vziať najlepšie vlastnosti Angularu a vytvoriť z nich nástroj, ktorý mu zjednoduší a zefektívni prácu.

Frontend aplikácie je implementovaný pomocou tohto frameworku. V sekcii 3.3.3 sa nachádza hlbší a podrobnejší popis jednotlivých vlastností frameworku.

2.2.3 Angular

Angular[5] je JavaScriptový open-source framework, vytvorený spoločnostou Google. Súčasná verzia sa označuje aj ako Angular 2+, ktorá má zatiaľ jedenásť podverzií[1]. Angular funguje na báze TypeScriptu, o ktorom si povieme viac v kapitole 3.2.

2.2.4 React

React[6] je frontendová JavaScriptová open-source[7] knižnica pre tvorbu užívateľských rozhraní. Na rozdiel od podobných frameworkov ako je Angular, React sa sústreďuje iba na jednu špecifickú oblasť `MVC`[8] architektúry, a to je vrstva pohľadu (anglicky `view`). Knižnica bola predstavená v roku 2013 spoločnosťou Facebook, ktorá ju už niekoľko rokov predtým používala a vyvíjala[5].

2.3 Backend

Backend, mnohokrát nazývaný aj ako server, je základom aplikácie. Tvorí časť, ktorá je skrytá pred používateľom. Jeho úlohou je prijímanie a posielanie dát pre klienta cez aplikačné rozhranie. Tieto údaje spracuje, ukladá, alebo modifikuje v databáze.

2.3.1 Node.js

Node.js[9] je spúšťacie prostredie(`runtime environment`) pre aplikácie napísané v jazyku JavaScript, mimo webového prehliadača. Je zakladaný na motore Chrome V8 JavaScript, čo znamená, že sa prostredie správa rovnako ako webový prehliadač Google Chrome[13]. Node.js sa najčastejšia využíva pri tvorbe serverovej časti webových aplikácií.

Platforma obsahuje viacero frameworkov pre uľahčenie vývoja aplikácií, najpopulárnejším z nich je `Express.js`[10], pomocou ktorého je implementovaná serverová časť aplikácie. Podrobnejší popis vlastností oboch technológií sa nachádza v sekcii 3.4.1.

2.3.2 Firebase

Firebase[11] je platforma typu backend-ako-služba(`Backend-as-a-Service`) , ktorá bola vytvorená v roku 2011. V roku 2014 bola odkúpená spoločnosťou Google, ktorá ju doteraz vyvíja[3]. Platforma pomáha v tvorbe, vylepšovaní a škálovaní aplikácií.

[5] https://angular.io/
[6] https://reactjs.org/
[7] Open-source software má dostupný zdrojový kód pre verejnosť
[8] Model-View-Controller - softwarová architektúra, ktorá rozdeľuje aplikáciu na dátový model, užívateľské rozhranie a logiku
[9] https://nodejs.org/en/
[10] https://expressjs.com/
[11] https://firebase.google.com/

BaaS je typ cloudových služieb pre tvorbu aplikácií bez investovania do vlastnej backend infraštruktúry. Uľahčuje vývojárom prácu zapúzdrením služieb na backendu, aby sa mohli sústrediť hlavne na tvorbu frontendu. Medzi tieto služby patrí autentifikácia užívateľov, manažment databáze, monitorovanie a analýza, vzdialené aktualizácie, cloudové úložisko alebo hosting. Všetky tieto služby Firebase poskytuje.

2.3.3 PHP

Hypertextový preprocessor[12], v skratke PHP[12], je populárny skriptovací jazyk, ktorý beží na strane servera. Slúži na vytváranie statických aj dynamických web stránok a aplikácií. Jazyk je bezplatný a použiteľný medzi viacerými platformami. Na PHP sa zakladá viacero frameworkov, ktoré poskytujú základnú štruktúru pre zjednodušený a zrýchlený vývoj. Medzi najpopulárnejšie patria Laravel a Symfony.

Laravel[13] je najpopulárnejší webový PHP framework. Jeho použitie je bezplatné a má voľne dostupný zdrojový kód pre verejnosť. Laravel sa stal obľúbeným kvôli jeho rýchlosti a jednoduchosti. Umožňuje vytváranie komplexných webových aplikácií, bez ohľadu na ich rozsiahlosť. Podporuje MVC architektúru, poskytuje smerovanie, autentifikáciu užívateľov, a mnoho ďalších[12].

Symfony[14] je jeden z najstarších a najspoľahlivejších PHP frameworkov. Je výbornou voľbou pre vysoko škálovateľné projekty. S Laravelom majú veľa podobných vlastností. Kým Symfony sa skôr zameriava na pokročilých programátorov, Laravel je jednoduchší pre nových vývojárov[12].

2.3.4 Python

Python[15] je interpretovaný, objektovo-orientovaný programovací jazyk[14]. Poskytuje dynamické typovanie premenných, ktoré je podrobnejšie rozpísané v sekcii 3.2, a rýchly vývoj aplikácií. Python patrí medzi modulárne jazyky, podporuje moduly a balíky, pomocou ktorých umožňuje znovupoužitie zdrojového kódu. Jazyk obsahuje viacero frameworkov, jedným z najpopulárnejších je Django.

Django[16] je open-source webový framework pre vytváranie webových aplikácií. Je postavaný na princípe MTV (Model-Template-View). Obsahuje dátový model a pohľad, ktorý je posielaný do šablóny. Namiesto písaní SQL dotazov preferuje prístup ORM[17], transformáciu Python objektov na dotazy.

2.4 Nástroje pre tvorbu prezentácií

Nástroje pre tvorbu prezentácií, ďalej označované ako slideshow frameworky, sú pomôcky pre vytváranie prezentácií vo webovom prehliadači pomocou webových technológií, ako HTML, CSS a JavaScript.

[12] https://www.php.net/
[13] https://laravel.com/
[14] https://symfony.com/
[15] https://www.python.org/
[16] https://www.djangoproject.com/
[17] ORM - Objektové relačné mapovanie

2.4.1 Reveal.js

Reveal.js[18] je najpopulárnejším nástrojom pre tvorbu prezentácií v prehliadači pomocou webových technológií. Framework je založený na čistom JavaScripte a nevyžaduje žiadny iný framework pre jeho použitie. Zdrojový kód projektu je voľne dostupný pre verejnosť.

Prezentovanie prezentácií v tejto práci sa rieši práve cez tento nástroj, jeho podrobnejší popis a príklad použitia sa nachádza v sekcii 3.3.6.

2.4.2 Eagle.js

Eagle.js[19] je slideshow framework založený na Vue.js. Podporuje animácie a motívy. Poskytuje užívateľom zopár komunitou vytvorených šablón pre rýchlejšiu tvorbu prezentácií.

Nevýhodou Eagle.js je, že jeho vývoj je zanedbaný. Posledná aktualizácia bola vydaná 23.10.2019. Framework používa takzvané mixiny, ktoré sú už v najnovšej verzii Vue.js neschválené (anglicky deprecated) a nahradené `Composition API`. O mixinoch a Composition API sa čitateľ dozvie viac v sekcii 3.3.4.

2.4.3 Impress.js

Impress.js[20] je určený hlavne pre upútanie pozornosti pomocou pestrých animácií. Silnou stránkou tohto frameworku sú animácie poskytované CSS3. Nástroj je vhodný pre prezentácie s malým obsahom.

[18] https://revealjs.com/
[19] https://github.com/zulko/eagle.js/
[20] https://impress.js.org/

Kapitola 3

Použité webové technológie

V tejto kapitole sa čitateľ oboznámi s použitými webovými technológiami v aplikácii. Na začiatku kapitoly sa uvádza popis zvolenej databázy. Po databáze sa nachádza krátky úvod do TypeScriptu a typových systémov. Ďalej sa v kapitole vysvetľuje myšlienka za zvolenie CSS preprocessora a CSS frameworku. Keďže aplikácia je implementovaná vo Vue.js, kapitola obsahuje jej podrobnejší popis a popis zvolených frontend technológií. Na záver sa čitateľ zoznámi s backend technológiami aplikácie a s architektúrou aplikačného rozhrania.

3.1 Databáza

V tejto sekcii sa čitateľ zoznámi s vlastnosťami NoSQL databáze MongoDB a s pomocnou knižnicou pre modelovanie objektov Mongoose.

3.1.1 MongoDB

MongoDB je open-source nerelačná databáza založená na dokumentoch[10]. Namiesto používania tradičných tabuliek a riadkov relačnej databázy, MongoDB vytvára kolekcie, ktoré sa skladajú z dokumentov vo formáte JSON. Jednotlivé dokumenty sa skladajú z párov kľúč-hodnota. Každý dokument môže mať rozlišný počet položiek. Veľkosť a obsah jednotlivých dokumentov sa môže taktiež od seba líšiť. Štruktúra dokumentu sa viac podobá štruktúre, ktorú vývojári vytvárajú pri programovaní tried a objektov.

3.1.2 Mongoose.js

Mongoose.js[11] je Node.js knižnica pre dátové modelovanie objektov pre MongoDB. Uľahčuje užívateľom prácu s MongoDB prekladaním dokumentov z databázy na objekty. Spravuje vzťahy medzi dátami, poskytuje validáciu schémy, typovanie a používa sa pre modelovanie dátových modelov na MongoDB objekty. Poskytuje dopytovacie funkcie nad modelmi, pomocou ktorých sa získavajú a ukladajú dáta do databázy. Tieto funkcie sú reťazovateľné(chainable).

Mongoose používa schémy pre modelovanie a správu dát v MongoDB. Schéma popisuje atribúty jednotlivých položiek s ktorými aplikácia pracuje, napríklad jeho dátový typ, či je položka povinná, alebo voliteľná.

3.2 TypeScript

Aby sa porozumela logika a nápad za vytvorením TypeScriptu a použitím ho v tomto projekte, je nutné sa najprv zoznámiť s nevýhodami JavaScriptu. JavaScript umožní zo statického webu vytvoriť dynamické aplikácie, ktoré používame dodnes[7]. Písať zdrojový kód čisto v JavaScripte je pomerne náročné a pri väčších projektoch takmer nemožné. Samotný jazyk neumožňuje počas programovania uvádzať dátové typy, čo veľmi sťažuje nápovedu v kóde a taktiež jeho automatickú kontrolu.

TypeScript bol vytvorený v roku 2012 firmou Microsoft[7]. Jeho zdrojový kód je voľne dostupný pre verejnosť. Jedná sa o nadstavbu jazyka JavaScript. Rozširuje ho o triedy, rozhrania, statické typovanie a ďalšie funkcionality z objektovo orientovaného programovania. TypeScript je využívaný aj spoločnosťou Googlu, ktorá ho integrovala do JavaScriptového frameworku Angular.

Ako už bolo spomenuté, TypeScript umožňuje programátorom statické typovanie. Existujú dva základné typové systémy:

- Pri **dynamickom typovaní** premenná má nastavený dátový typ, ale systém typ skrýva a nedáva ho vôbec najavo. Premenné pri dynamickom typovaní mnohokrát ani nemusia byť deklarované, akonáhle sa do nejakej premennej niečo uloží a jazyk zistí, že nebola nikdy deklarovaná, sám ju založí. Do tej istej premennej sa môže ukladať reťazec, potom objekt a potom pole. Jazyky využívajúce dynamický typovanie vnútorne automaticky menia dátový typ. Medzi tieto jazyky patrí Python, alebo práve JavaScript, v ktorých je vývoj rýchlejší vďaka menšiemu množstvu kódu.

- Pri **statickom typovaní** je striktne vyžadované definovanie typu premennej a typ je naďalej nemenný. Pri uložení hodnoty iného typu do premennej, kompilácia zdrojového kódu skončí chybovým hlásením.

TypeScript[7] je typizovaným jazykom, všetky premenné musia byť najprv deklarované s priradeným dátovým typom. Nevýhodou je, že kvôli deklaráciám je zdrojový kód objemnejší a vývoj pomalší. Naopak obrovskou výhodou je, že kompilátor pred spustením skontroluje, či všetky nastavené hodnoty vyhovujú ich dátovým typom. Dynamické typovanie síce vyzerá ako výhodné, ale pri čítaní zdrojového kódu je ťažké zistiť typ hodnoty premennej. TypeScript do základnej miery taktiež povoľuje využitie dynamických typov cez takzvané `union typy`. Union typy umožňujú nastavenie viacero dátových typov premennej, napríklad `let data: string | number`, kde premenná `data` môže obsahovať reťazec a aj číslo. TypeScript pri priradení hodnoty nesprávneho dátového typu nedovolí program ani skompilovať a na chybu ihneď upozorní.

3.3 Frontend

Sekcia uvádza jazyky a technológie použité pri implementácii klientskej časti aplikácie. Popisuje základné jazyky, ako je HTML a CSS, preprocessor Sass a framework užívateľského rozhrania Buefy. Ďalej v sekcii sa nachádza podrobnejší popis frameworku Vue.js, nadstavby Nuxt.js a technológie Composition API. Záver sekcie obsahuje krátky úvod do slideshow frameworku Reveal.js a príklad jeho použitia.

3.3.1 HTML

Hypertextový značkovací jazyk, alebo skrátene HTML je jazyk vytvorený Timom Berners-Leeom v roku 1991[2]. Hneď na začiatku treba zdôrazniť, že sa nejedná o programovací jazyk. HTML neumožňuje vytváranie premenných, alebo funkcií. Jazyk je určený na vytváranie webových stránok a na organizovanie, formátovanie rôznych informácií zobraziteľných vo webovom prehliadači. Umožňuje užívateľom vytvárať a štrukturovať sekcie, nadpisy, paragrafy, odkazy a mnoho ďalších pre webové stránky a aplikácie. Pri práci s HTML sa používajú štruktúry, ktoré sa nazývajú značky `(tagy)` a `atribúty`, pomocou ktorých sa štrukturujú jednotlivé časti web stránky.

3.3.2 CSS

Kaskádové štýly, alebo skrátene CSS je rozšírenie jazyka HTML. Jazyk bol vytvorený a je udržiavaný skupinou v organizácii W3C[1]. Prvá verzia vznikla v roku 1996 a obsahovala iba modifikácie písmen, okrajov a farieb. Súčasná verzia je `CSS3`[8]. Jazyk umožňuje vizuálne formátovanie internetových dokumentov a špecifikuje, ako sa majú dokumenty prezentovať pre užívateľa. Pomáha v štýlovaní jednotlivých elementov a v upravovaní ich umiestnenia. Využíva sa od základných štýlov, ako zmena farby textu, alebo veľkosti až k rozsiahlejším zmenám rozloženia a aj pre zobrazovanie animovaných efektov.

V súčasnej dobe sa často používajú preprocesory pri práci s CSS. V aplikácii sa využíva preprocesor `Sass`[2]. Preprocesor je program, ktorého výstup je spracovaný ďalším programom, t.j. štýly napísané v Sass sú spracované ďalším programom, z čoho vznikne CSS[6]. Hlavným dôvodom ich použitia je prehľadnosť kódu a súborov. Medzi najpopulárnejšie preprocesory patrí Sass, Less a Stylus.

Sass

Sass je najstarším CSS preprocesorom[6]. Je ľahko naučiteľný a pomerne jednoduchý jazyk. Medzi jeho hlavné výhody patrí vytváranie premenných a uchovávanie hodnôt do nich, vytváranie vnorených štýlov, matematických operácií, cyklov a funkcií. Sass naďalej podporuje importovanie externých súborov, čo umožňuje rozdelenie štýlov do rôznych logických častí. Jednotlivé celky sa preprocesorom následne skompilujú do jedného CSS súboru.

Buefy

Buefy[3] poskytuje predpripravené komponenty pre užívateľské rozhranie. Uľahčuje prácu vývojárom so štýlovaním základných elementov dokumentu. V aplikácii sa využíva komponenta pre tlačidlá, skrývanie obsahu, dialógy, kolónky vstupu užívateľa, vykresľovanie ikon, animácia načítania, navigačný panel a bočný panel. Zdrojový kód frameworku je voľne dostupný pre verejnosť. Každá komponenta je responzívna, sama sa prispôsobuje rôznym rozlíšeniam obrazovky a týmto uľahčuje vývoj aj pre mobilnú platformu.

Framework je vytvorený vo Vue.js a zakladá sa na CSS knižnici Bulma[4]. Bulma je taktiež open-source knižnica, ktorá je postavená na CSS architektúre `flexbox`. Samotná knižnica poskytuje CSS triedy pre štýlovanie HTML dokumentu. Buefy pomocou týchto

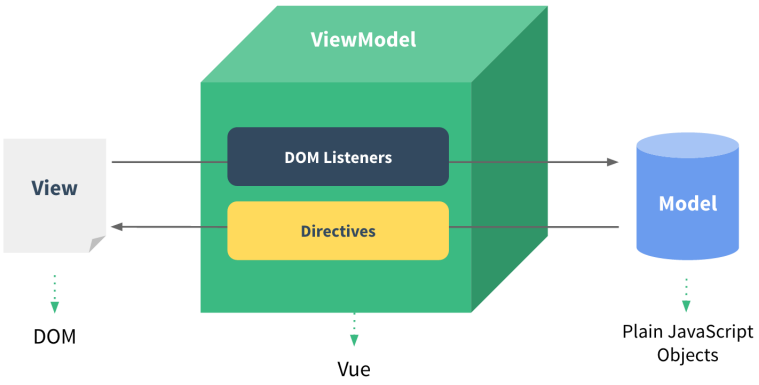
[1] Skupina ľudí, ktorá navrhuje vylepšenia a špecifikácie o Internete a jeho vývoji.
[2] https://sass-lang.com/
[3] https://buefy.org/
[4] https://bulma.io/

tried vytvorí jednotlivé komponenty užívateľského rozhrania pre Vue.js, ktoré sú potom jednoducho použiteľné v aplikácii. Tieto komponenty sa označujú predponou `b-`, napríklad `<b-button></b-button>`.

3.3.3 Vue.js

Ako už bolo spomínané, Vue.js patrí medzi najpopulárnejšie nástroje pre tvorbu jednostránkových aplikácii. Veľkou výhodou frameworku je, že patrí medzi progresívne frameworky, čo znamená, že aplikáciu vieme rozdeliť na viacero častí, ktoré môžu byť samostatne vyvíjané.

Dizajn Vue.js sa inšpiroval MVVM[5] architektúrou[15]. Zameriava sa na vrstvu pohľadu. Spája pohľad a model cez dvojstrannú väzbu (two-way binding), ako je vidieť na obrázku 3.1. Ostatné funkcionality jednostránkovej aplikácii, ako smerovanie a úložisko(ďalej označované aj ako `store`) môžu byť doplnené cez pomocné knižnice.



Obr. 3.1: Znázornenie úlohy Vue.js v MVVM architektúre [15].

Inštalácia

Vue.js bol vytvorený, aby mohol byť ľahko osvojiteľný a použiteľný. Môže byť začlenený do projektu viacerými spôsobmi. Štyri hlavné spôsoby inštalácie sú nasledovné:

- Importovanie balíčka do HTML stránky cez CDN[6]:

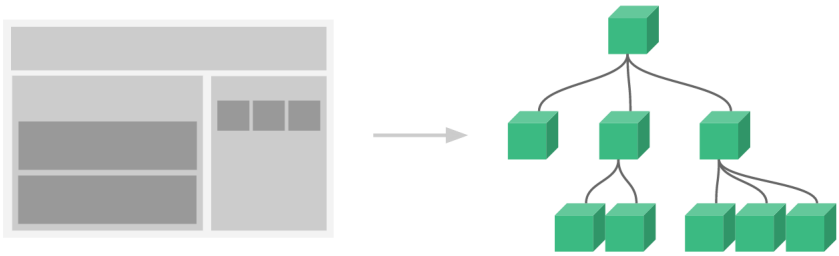
  ```
  <script src="https://unpkg.com/vue@next"></script>
  ```

- Stiahnutie JavaScriptových súborov a importovanie ich rovnakým spôsobom ako pri CDN.
- Inštalácia cez správcu balíčkov npm.
- Použitie oficiálneho CLI[7] pre zkonštruovanie projektu.

[5] MVVM - Model-View-ViewModel architektúra
[6] Content Delivery Network - server pre doručovanie obsahu, skriptov, alebo knižníc.
[7] Command Line Interface - príkazový riadok

Komponenty

Komponenty hrajú vo Vue.js veľmi dôležitú rolu, umožňujú tvorbu vysoko škálovateľnej aplikácie, ktorá je poskladaná z malých a znovupoužiteľných častí. Môžeme mať komponenty napríklad pre hlavičku, bočný panel, obsah, ktoré môžu obsahovať ďalšie komponenty pre zoznam prezentácií a výpis detailu prezentácie. Výsledkom rozdelenia do jednotlivých častí je ľahko udržateľný a prehľadný zdrojový kód. Komponenty tvarujú hierarchiu vnoreného stromu, ktorá je znázornená na obrázku 3.2. Tento strom reprezentuje aplikačné rozhranie.



Obr. 3.2: Znázornenie komponentov ako vnorený strom[15].

Registrovanie komponentov

Existujú dva postupy registrovania jednotlivých komponentov: `lokálne` a `globálne`. Lokálne komponenty je potrebné pred každým použitím importovať, na rozdiel od globálnych, ktoré stačí zaregistrovať raz a môžu byť použité v šablóne hociktorej komponenty.

Typický príklad pre globálnu komponentu je dizajnový prvok, ako napríklad animácia pri načítaní údajov, označovaný ako spinner. Tento typ animácie sa používa pri každom načítaní údajov, na viacerých miestach v aplikácii. Aby sme ho nemuseli pri každom použití importovať, stačí keď si ho raz globálne zaregistrujeme a môžeme ho voľne používať.

Príklad lokálnej komponenty je formulár pre registrovanie užívateľa, ktorý sa pravdepodobne použije iba raz a nepotrebujeme mať k nej prístup v celej aplikácii.

Komunikácia medzi komponentami

Komponenty začnú byť ešte viac užitočné pri vymieňaní dát medzi sebou. Štandardne, každá jedna komponenta je izolovaná, čo znamená, že nemá prístup k rodičovským údajom. Majme komponentu pre stručný výpis informácií o prezentácii, po kliknutí na ňu sa zobrazí ďalšia komponenta s detailným výpisom. Pri tomto príklade potrebujeme predať informácie o prezentácii z jednej komponenty na druhú. Na komunikáciu medzi rodičom a potomkom sa používa `props` a metóda `$emit`.

Props sú atribúty, ktorým vieme nastaviť hodnoty. Slúžia na predanie údajov z rodiča na potomka. Po nadobudnutí hodnoty sa z nich stanú premenné použiteľné v komponente. Počet props nie je obmedzený a prijíma ľubovolný typ údajov.

Metóda $emit slúži na komunikáciu opačným smerom. Pomocou tejto metódy vieme vytvoriť vlastnú udalosť, ktorú rodič môže odpočúvať. Majme komponentu potomka v ktorej

sa nachádza tlačidlo. Kliknutím na tlačidlo sa zavolá metóda $emit do ktorej sa predá meno udalosti a voliteľné dáta. Rodičovská komponenta môže odchytiť túto udalosť a reagovať na ňu. Vysielanie udalostí je užitočné, keď máme viacero potomkov s rovnakou logikou, týmto spôsobom nemusia všetky tieto komponenty obsahovať tú istú logiku, stačí keď bude implementovaná iba v rodičovskej komponente a pristúpi sa k nej cez $emit.

Životný cyklus komponenty

Vo Vue.js každá komponenta je samostatná Vue inštancia[8] a každá inštancia má svoj životný cyklus. Životný cyklus komponenty sa skladá z niekoľkých krokov, ako napríklad jej vytvorenie, pripevnenie inštancie k DOM[9], aktualizácia DOM-u pri zmene údajov, alebo jej zrušenie. Vývojári majú pri každom kroku možnosť pridania vlastného kódu cez `funkcie životného cyklu(lifecycle hooks)`. Medzi najpoužívanejšie patria:

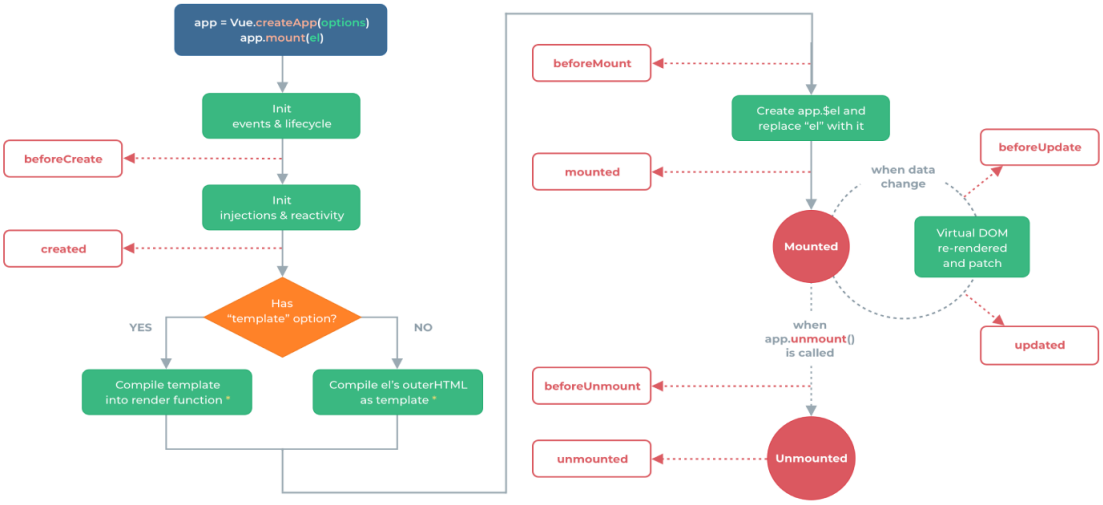
- **beforeCreated** - volá sa hneď po inicializácii inštancie, ktorá zatiaľ nič neobsahuje.
- **created** - pri tomto kroku sa dokončilo nastavenie pozorovania dát, computed premenných, metód a udalostí. Zatiaľ nemáme prístup k DOM. Používa sa pre registrovanie vlastných udalostí.
- **beforeMount** - volá sa pred pripevnením DOM.
- **mounted** - najpoužívanejšia funkcia životného cyklu. Volá sa po pripevnení inštancie k DOM. V tejto funkcii už máme prístup k jednotlivým elementom v DOM. Častokrát používaná pre sťahovanie dát z API.
- **beforeUpdate** - volá sa pri zmene dát. Poskytuje prístup k existujúcemu DOM pred jeho aktualizáciou.
- **updated** - funkcia volaná po znova-vykreslení DOM.
- **beforeUnmount** - volá sa pred odpojením inštancie od DOM-u.
- **unmounted** - po tomto kroku je inštancia, jej potomkovia a aj udalosti kompletne odpojené. Používaná pre odpojenie vlastných udalostí.

Pri použití `Composition API`, o ktorom si povieme viac v sekcii 3.3.4, sa tieto funkcie trochu líšia. Funkcia životného cyklu `beforeCreated` a `created` sú súčasťou funkcie `setup()` a pred každú funkciu je potrebné pridať predponu **on**: **on**Created, **on**Mounted, atď.

Šablóna

Komponenta sa vykresľuje cez HTML šablónu. Okrem základnej funkcionality jazyka HTML umožňuje výpis statických aj dynamických hodnôt, dynamickú aktualizáciu CSS štýlov a tried, použitie podmienok na obmedzenie vykresľovania a použitie cyklov pre prechádzanie hodnôt v poli. V šablóne má vývojár prístup k premenným aj metódam v komponente. Obsah šablóny sa musí nachádzať medzi značkami `<template></template>`.

[8] Jeden konkrétny exemplár triedy
[9] Objektový model HTML dokumentu



Obr. 3.3: Graf životného cyklu komponenty[15].

3.3.4 Composition API

Vue.js verzia 2 priniesla funkcionalitu mixinov. Mixiny boli hlavným nástrojom pre vyčlenenie jednotlivých častí logiky komponenty do znovupoužiteľných celkov. Umožňujú tvorbu premenných, funkcií a logiky vo funkciách životného cyklu, rovnakým spôsobom ako v komponentoch. Mixiny avšak spôsobujú viacero konfilktov. Pri ich importovaní sa premenné a funkcie z mixinu zlúčia do súboru komponenty a môžu vzniknúť konflikty rovnakého pomenovania. Ďalším významným problémom je limitovaná znovupoužiteľnosť. Mixiny nepodporujú predávanie parametrov pre zmenu logiky, tým pádom majú zredukovanú flexibilitu.

Všetky tieto problémy sa nástupom Composition API vyriešili. Composition API rovnako ponúka vyčlenenie častí logiky a ich znovupoužitie. Avšak neslúži iba pre znovupoužiteľnosť, vyčleniť sa môže aj logika, ktorá v komponente zaberá príliš veľa miesta. Výsledkom je prehľadnejšia komponenta, v ktorej sa jednoduchšie orientuje. Súbor, ktorý obsahuje vyčlenenú logiku sa označuje ako `skladateľný(composable)`. Composition API poskytuje vo vyčlenených súboroch rovnaké funkcionality ako v komponentoch, možnosť deklarovania reaktívnych premenných, computed premenných, funkcií, sledovanie premenných, prístup ku kontextu aplikácie a k funkciám životného cyklu.

Začiatočným bodom v Composition API je funkcia `setup`. Setup funkcia sa vykoná pred vytvorením komponenty, hneď ako sa atribúty `props` spracujú. Funkcia prijíma props a kontext aplikácie ako parameter. Všetko, čo sa z funkcie vracia je dostupné v komponente a aj v šablóne komponenty.

Reaktívne premenné

Composition API umožňuje vytváranie reaktívnych premenných pomocou funkcie `ref`. Reaktívne premenné slúžia na automatické prekreslenie časti šablóny pri ich zmene. Príklad zápisu:

```
const currentSlide = ref<number>(1)
```

Ref zoberie danú hodnotu a zabalí ju do `Proxy` objektu, pomocou ktorej Vue reaguje na zmeny. K samostatnej hodnote objektu sa pristupuje cez atribút `.value` a cez rovnaký atribút sa taktiež hodnota upravuje. Kvôli tomu sa premenná deklaruje ako konštanta, lebo pri modifikácii sa upravuje hodnota v objektu a nie celý objekt. V šablóne sa k hodnote pristupuje rovno cez objekt a nie je potrebné použiť atribút `.value`.

Composition API ponúka taktiež funkciu `reactive` pre reaktívny zápis premenných. Príklad použitia:

```
const presentation = reactive<Presentaion>({
        title: "Moja prezentácia",
        slides: ["Stránka 1", "Stránka 2"]
})
```

Funkcia zoberie objekt a každú jednu položku priradí do Proxy objektu. Logika za funkciou je pomerne rovnaká ako pri `ref`, rozdiel je pri prístupe k jednotlivým položkám a ich hodnotám, kde pri `reactive` nie je potrebné uviesť atribút `.value`. Pri použití v šablóne je potrebné uviesť objekt aj atribút, napríklad `presentaton.title`. Tento zápis sa dá zjednodušiť pomocou funkcie `toRefs`. Táto funkcia vyžaduje reaktívny objekt ako parameter a používa sa pri vrátení hodnôt vo funkcii `setup`. Funkcia `toRefs` zoberie objekt a vráti každý jeden atribút ako samostatný `ref` objekt, týmto spôsobom sa k premennej v šablóne pristupuje iba cez atribút, napríklad `title`. Pomocou `reactive` vieme jednotlivé premenné, ktoré súvisia so sebou zoskupiť do jedného objektu pre prehľadnejšiu orientáciu v zdrojovom kóde.

Ako je v hore uvedených príkladoch vidieť, obe reaktívne funkcie podporujú typovanie pomocou TypeScriptu. Typ sa uvádza medzi zátvorky `< >`.

3.3.5 Nuxt.js

Nuxt.js je nadstavbou Vue.js. Framework zjednodušuje vývoj univerzálnych a jednostránkových Vue aplikácií. Ako už bolo spomenuté, Vue.js aplikácie sú jednostránkové, obsahujú iba jeden HTML dokument `index.html`. Obsah tohto súboru sa mení podľa potreby pomocou JavaScriptu. Tým pádom na serveri je dostupná iba jedna stránka aplikácie, ktorá je práve aktívna. Tento prístup zabraňuje internetovým vyhľadávačom(SEO) nájsť ostatné stránky aplikácie podľa vyhľadávanej frázy. Vyhľadávač je program(robot), ktorý hľadá stránky, alebo dokumenty, ktoré obsahujú vyhľadávané slovo, alebo celú frázu. Podľa takýchto robotov funguje aj najpopulárnejší internetový vyhľadávač Google. Nuxt.js poskytuje riešenie na tento problém vykresľovaním obsahu na strane servera, týmto uľahčuje optimalizáciu webu pre vyhľadávače(`SEO`).

Hlavným dôvodom použitia Nuxt.js pri tomto projekte je zjednodušený vývoj pre programátora. Framework poskytuje jednoduchú inštaláciu projektu, vytvorí základnú štruktúru aplikácie, zabezpečuje smerovanie a kompiláciu aplikácie pomocou Webpacku a Babela. Obsahuje moduly pre jednoduchú autentifikáciu užívateľa, úložisko, HTTP komunikáciu a mnoho ďalších. Pomocou týchto modulov sa dá vyhnúť nadbytočnému boilerplate kódu. Vývojárom poskytuje vývojové prostredie, ktoré sa automaticky aktualizuje pri zmene v zdrojovom kóde.

3.3.6 Reveal.js

Ako už bolo v práci spomínané, Reveal.js je nástroj pre vytváranie prezentácií v prehliadači pomocou webových technológií. Dokumentácia nástroja je veľmi pestrá a prehľadná. Frame-

work sa inštaluje cez správcu balíkov `npm`. Použitie frameworku je veľmi jednoduché, celková prezentácia musí byť v šablóne zabalené do bloku s triedou `.reveal`. Stránky prezentácie musia patriť pod blok s triedou `.slides` a obsah jednotlivých stránok musí byť v značke `<section>`. Výsledná prezentácia je interaktívna a ovláda sa pomocou klávesových šípok. Jednoduchý príklad použitia:

```
<div class="reveal">
    <div class="slides">
        <section>Stránka 1</section>
        <section>Stránka 2</section>
    </div>
</div>
```

3.3.7 Ostatné balíčky a knižnice

auth-next

Auth-next je Nuxt.js modul pre autentifikáciu užívateľov. Implementácia autentifikácie pomocou tohto modulu sa nachádza v sekcii 5.2.1.

axios

Axios je HTTP klient pre prehliadač. Knižnica sa v aplikácii využíva pre komunikáciu medzi frontendom a backendom na klientskej časti komunikácie.

node-sass a sass-loader

Tieto knižnice slúžia pre načítanie Sass súborov a ich kompiláciu do jednotného CSS súboru.

vuedraggable

Balík poskytuje Vue komponentu pre jednoduché použitie funkcionality ťahaj-a-pusti(drag-n-drop).

vuex-persistedstate

Knižnica slúži na perzistovanie stavov Vuex úložiska medzi znova načítaním stránky. Využitie knižnice je popísané v sekcii 5.2.3.

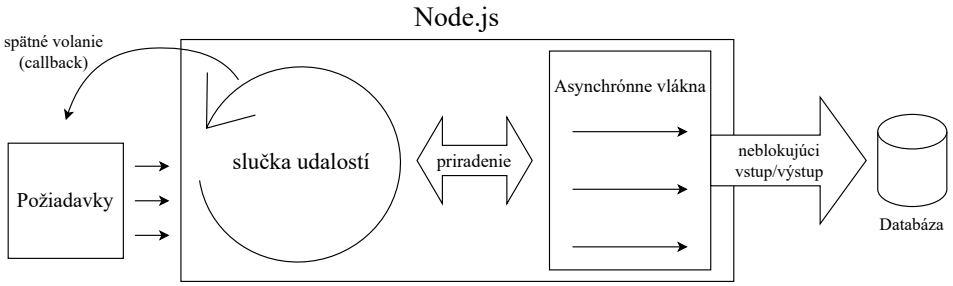
3.4 Backend

Táto sekcia oboznamuje s výhodami spúšťacieho prostredia Node.js a krátky popis jeho frameworku Express.js. Sekcia ďalej obsahuje popis architektúry aplikačného rozhrania REST a potrebné požiadavky, aby API mohlo byť považované za RESTful.

3.4.1 Node.js a Express.js

Spúšťacie prostredie Node.js je poháňané asynchronnými udalosťami, ktoré umožňujú tvorbu vysoko škálovateľných a výkonných aplikácií, čo bolo primárnym dôvodom voľby tohto nástroja pre backend projektu.

Za vysokou výkonnosťou stojí jeho jadro, ktoré obsahuje slučku udalostí(event loop)[13]. Do tejto slučky sa priraďujú všetky požiadavky, ktoré sú nasledovne priradené k asynchronným vláknam. Pri Node.js sa netreba obávať kvôli uviaznutiam(deadlockom), kvôli funkcionalite `neblokujúci Vstup/Výstup`. Operácie, ako čítanie a zápis do databázy, alebo HTTP požiadavky sú riešené cez tieto udalosti, ktoré umožňujú paralelné spracovanie požiadaviek. Napríklad majme dvoch užívateľov, ktorí si naraz chcú zobraziť informácie o prezentácii. Pomocou neblokujúceho Vstupu/Výstupu užívateľ2 si môže požiadať o dáta z databázy bez toho, aby sa počkalo na spracovanie požiadavky užívateľa1. Celá architektúra Node.js je grafický znázornená na obrázku 3.4.



Obr. 3.4: Architektúra Node.js.

Express.js

Express.js je najpopulárnejším webovým frameworkom založeným na Node.js[4]. Je minimalistický a flexibilný. Poskytuje robustnú škálu funkcionalít pre tvorbu serverovej časti webových aplikácií a množstvo HTTP utilít pre rýchlu a jednoduchú tvorbu aplikačného rozhrania. Umožňuje spracovanie jednotlivých HTTP dotazov na rôzne URL adresy.

3.4.2 Aplikačné rozhranie

Aplikačné rozhranie je realizované pomocou architektúry REST. Skratka znamená Representation State Transfer, čiže reprezentačný prenos stavu. Architektúra umožňuje pristupovať k dátam a vykonávať nad nimi základné operácie, ako ich vytvorenie, čítanie, aktualizáciu a mazanie. Na prenos dát sa používajú metódy protokolu HTTP. Najčastejšie používané metódy sú:

- **GET** pre získanie dát.
- **POST** pre vytváranie dát.
- **DELETE** pre mazanie dát.
- **PUT** pre upravovanie dát.

Protokol ponúka aj metódy `HEAD`, `PATCH`, `OPTIONS`, ktoré avšak nie sú často používané.

Architektúra poskytuje pre každý jeden zdroj samostatný koncový bod, cez ktorý sa dajú dáta spravovať. Tabuľka koncových bodov použitých v aplikácii s ich popisom sa nachádza v sekcii 4.1.

Aplikačné rozhranie musí spĺňať nasledujúce požiadavky, aby bola považovaná za RESTful:

- **Klient-server architektúra** - cieľom je rozdeliť klientskú časť aplikácie od serverovej. Umožňuje nezávislý vývoj oboch častí aplikácie. Užívateľské rozhranie na viacerých platformách môže mať rovnaký backend.
- **Bezstavová architektúra** - požiadavky na server musia obsahovať všetky potrebné informácie na ich spracovanie. Stav sa uchováva na strane klienta a nie na strane servera.
- **Uchovávateľnosť v cache pamäti** - kvôli vysokému počtu požiadaviek, každá jedna požiadavka musí byť označená ako uložiteľná, alebo neuložiteľná do cashe pamäte. Výsledkom je zefektívnená výkonnosť servera.
- **Vrstovateľnosť** - rozdelenie aplikácie do jednotlivých vrstiev, kvôli škálovateľnosti. Každá vrstva má svoj vlastný účel.
- **Jednotné rozhranie** - základ architektúry, musí byť zadefinovaný spôsob komunikácie medzi klientom a serverom.
- **Code on Demand** - voliteľná požiadavka architektúry. Umožňuje posielanie zdrojového kódu pre klienta, čím rozšíri jeho funkcionalitu.

3.4.3 Ostatné balíčky a knižnice

bcrypt

Bcrypt je knižnica pre Node.js, ktorá umožňuje hašovanie hesiel.

cors

Knižnica poskytuje middleware pre Express.js, ktorý povoluje zdieľanie zdrojov medzi inými doménami(cross-origin resource sharing).

day.js

Day.js je minimalistická JavaScript knižnica pre prácu s časom a dátumom.

dotenv

Modul načíta premenné zo súboru prostredia `.env` a sprístupní ich cez objekt `process.env`.

jsonwebtoken

Knižnica implementuje autentifikáciu užívateľa cez JSON Web Token. Implementácia autentifikácie pomocou tejto knižnice je podrobnejšie popísana v sekcii 5.3.3.

ts-node, tslint a typescript

Knižnice ts-node a typescript slúžia pre spustenie prostredia Node.js s podporou pre jazyk TypeScript. Knižnica tslint je nástroj pre analýzu zdrojového kódu v jazyku TypeScript.

nodemon

Nodemon je nástroj pre vytvorenie vývojového prostredia Node.js aplikácie, ktorý automaticky aktualizuje aplikáciu pri detekovaní zmien v zdrojových súboroch. Vývojové prostredie sa vytvorí pomocou príkazu v termináli:

```
nodemon ts-node ./src/server.ts
```

Kapitola 4

Návrh riešenia

V tejto kapitole sa čitateľ zoznámi s návrhom celej aplikácie. V prvej sekcii sú rozobrané prípady použitia aplikácie. Ďalej sa čitateľ dočíta o návrhu architektúry systému, o návrhu užívateľského rozhrania, o návrhu databázy a na záver o návrhu serverovej aj klientskej časti aplikácie.

4.1 Použitie aplikácie

Pri príchode na webovú stránku sa užívateľ ocitne na autentifikačnej stránke, kde sa musí prihlásiť, alebo zaregistrovať, aby mohol postúpiť ďalej.

Po úspešnom prihlásení užívateľ je presmerovaný na domovskú stránku. Na domovskej stránke má prístup k zoznamu vlastných prezentácií, ktoré už vytvoril. Vie si tu vytvoriť novú prezentáciu kliknutím na tlačidlo `Create New`, alebo sa odhlásiť tlačidlom `Logout`. Zoznam obsahuje kartičky jednotlivých prezentácií so stručným popisom informácií. Kartička obsahuje tlačidlo `Delete` pre odstránenie celej prezentácie. Kliknutím na kartičku sa zobrazí detailný popis prezentácie. V detaily má užívateľ prístup k vytvoreným verziám a popisom k nim. Nachádza sa tu náhľad do stránok jednotlivých verzií pre uľahčenie voľby správnej verzie. Užívateľ si vie zobraziť prezentáciu v prezentačnom móde cez tlačidlo `View`, upraviť verziu cez tlačidlo `Edit`, odstrániť verziu tlačidlom `Delete v_` a stiahnuť prezentáciu vo formáte PDF tlačidlom `Download`.

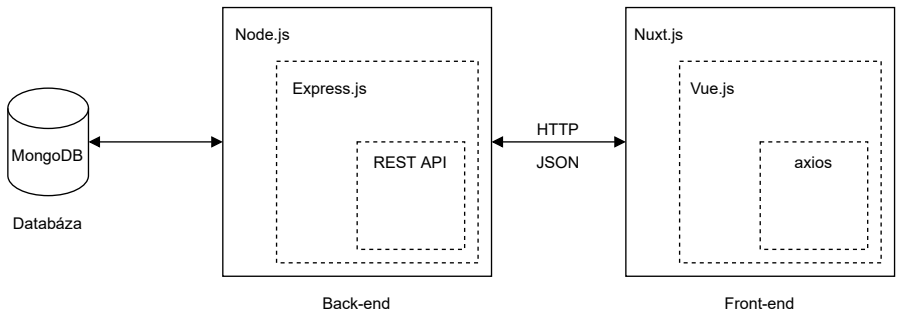
Pri vytváraní novej prezentácie, alebo upravovaní už existujúcej nás aplikácie presmeruje na stránku editora. Editor obsahuje mnoho užitočných nástrojov pre zjednodušenie tvorby pomocou jazyka Markdown, ako napríklad typografické nástroje, vloženie zoznamov, tabuliek, obrázkov, odkazov a častí zdrojového kódu. Veľkou výhodou je výskyt náhľadu v editore, kde je hneď vidieť sformátovaný obsah stránky. V bočnom paneli má užívateľ možnosť sa vrátiť na domovskú stránku cez tlačidlo `Home`. Po kliknutí na tlačidlo `Save` sa zobrazí panel na uloženie prezentácie. Pri upravovaní prezentácie aplikácia ponúka úpravu názvu a pridanie popisu k verzii. Prezentáciu vie tvorca uložiť pod novou verziou, alebo môže aktualizovať práve upravovanú verziu. Kliknutím na tlačidlo `Preview` sa aplikácia prepne do prezentačného módu. Prepínanie stránok v prezentačnom móde funguje pomocou klávesnicových šípok.

Tvorca si vie zobraziť panel so stránkami cez tlačidlo `Slides`. Panel obsahuje zoznam jednotlivých stránok prezentácie. Ich poradie môže byť upravené pomocou technológie ťahaj-a-pusti(drag-and-drop). Užívateľ má možnosť prepnutia zoznamu stránok do mriežkového pohľadu(grid view), kde sa dá jednoduchšie orientovať a usporiadať jednotlivé

stránky. Aplikácia umožňuje kopírovanie stránok medzi prezentáciami jednoduchým kliknutím na tlačidlá `Kopírovať` a `Prilepiť`.

4.2 Architektúra aplikácie

Aplikácia je rozdelená na dve časti, na klientskú a serverovú. Jedná sa o takzvanú dvojvrstvovú architektúru `klient-server`. Klientská časť sa nazýva aj ako prezentačná vrstva. Poskytuje interaktivitu užívateľom, má za úlohu zobrazovanie obsahu a údajov získaných zo servera. Zbiera užívateľom zadané dáta cez rôzne vstupy. Tieto dáta môžu byť spracovávané a upravované už na klientovi a sú posielané na server cez protokol HTTP[1] pomocou knihovne *axios*. Server údaje prijíma a ďalej spracováva. Medzi úlohy serverovej časti patria: prijímanie požiadavkov od klienta, posielanie odpovedí na požiadavky stavovými kódmi a údajmi, zabezpečenie a udržovanie komunikácie cez autentifikáciu a autorizáciu užívateľov, vykonávanie rôznych výpočtov, formátovanie údajov pre databázu, alebo klienta. Klient nekomunikuje priamo s databázou, na to slúži server. Server údaje od klienta upravuje, aby vyhovovali formátu v databáze. Taktiež si vypýta dáta z databázy, ktoré spracováva a posiela klientovi. Grafické znázornenie architektúry pre jednoduchšie pochopenie sa nachádza na obrázku 4.1.



Obr. 4.1: Znázornenie architektúry aplikácie.

4.3 Návrh frontendu

V aplikácii sa využíva framework Nuxt.js, ktorý je nadstavbou Vue.js. Nuxt.js uľahčuje vývoj jednostránkových aplikácií a poskytuje vývojárom lepší zážitok pri programovaní. Aktuálne stabilná verzia Vue.js a Nuxt.js pri začatí projektu je verzia 2, ale v aplikácii sa už využívajú `Composition API` vlastnosti verzie 3, ktoré je potrebné nainštalovať ako balík cez NPM a pridať medzi moduly Nuxt.js v konfiguračnom súbore.

4.3.1 Pohľad aplikácie

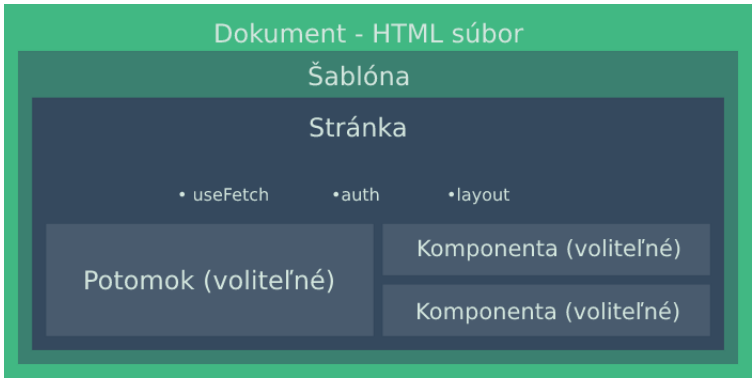
Pohľad v Nuxt.js aplikácii sa skladá z viacerých vrstiev. Najvyššia vrstva je HTML súbor `App.html`, ktorý je vytvorený automaticky a slúži pre vytvorenie kostry aplikácie. Tento súbor obsahuje všetok obsah, atribúty pre hlavičku a telo HTML dokumentu.

[1] Hypertextový prenosový protokol

Nižšia vrstva obsahuje šablónu aplikácie, pomocou ktorej sa zadefinuje rovnaký obsah na viacerých stránkach. Dobrým príkladom pre obsah v šablóne je navbar, ktorý môže byť dostupný na viacerých stránkach aplikácie.

Pod šablónou sa nachádzajú komponenty jednotlivých stránok aplikácie. Každá stránka je samostatná Vue komponentom, ku ktorým Nuxt.js prideľuje špeciálne atribúty a funkcie. Medzi ne patrí **useFetch**, funkcia pre asynchrónne sťahovanie dát hneď pri návšteve stránky, naďalej atribút **auth** vyžadujúci autentifikáciu pre navštívenie stránky, alebo atribút **layout** určujúci, ktorú šablónu má stránka využívať.

Každá stránka môže mať niekoľko potomkov. Potomok má rovnakú štruktúru ako rodičovská stránka a taktiež obsahuje Nuxtom pridelené atribúty a funkcie. Stránka sa môže skladať z viacerých komponentov, ich výhody už boli v tejto práci zmienené vyššie.

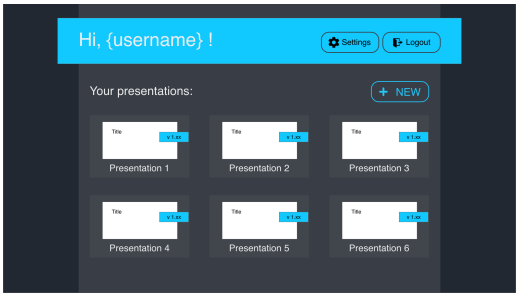


Obr. 4.2: Kompozícia pohľadu v Nuxt.js.

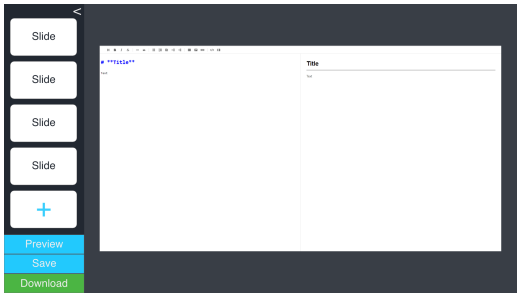
4.3.2 Užívateľské rozhranie

Užívateľské rozhranie je dôležitou súčasťou aplikácie. Je to časť, s ktorou užívateľ pracuje, a preto je dôležité, aby orientácia na nej bola čím jednoduchšia a prehľadnejšia. Cieľom bolo vytvoriť minimalistické rozhranie, na ktorom sa ľahko nájdu potrebné ovládacie prvky.

Prototyp rozhrania bol vytvorený pomocou návrhového programu Adobe XD. Prvý návrh domovskej stránky a editora je vidieť na obrázku 4.3 a 4.4. Odvtedy aplikácia prešla viacerými prestavbami a dizajn sa výrazne zmenil.



Obr. 4.3: Prototyp domovskej stránky



Obr. 4.4: Prototyp editora

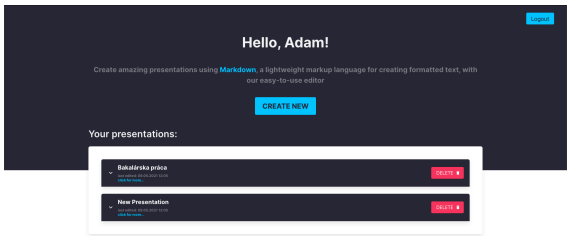
Autentifikačná stránka

Autentifikačná stránka je prvá stránka, na ktorej sa užívateľ ocitne pri prvom navštívení aplikácie. Stránka obsahuje prihlasovací panel, kde je nutné zadať správny e-mail a heslo. Pri novom užívateľovi je najprv nutná registrácia cez registračný panel. Aplikácia vyžaduje iba potrebné užívateľské údaje, ako meno, e-mail a heslo.

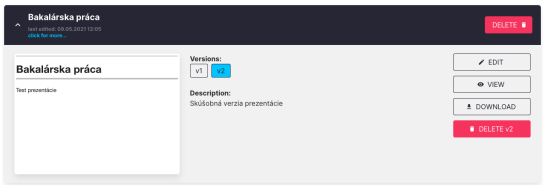
Domovská stránka

Po úspešnom prihlásení sa zobrazí domovská stránka. Domovská stránka by mala obsahovať všetky potrebné informácie pre užívateľa. Stránka je znázornená na obrázku 4.5. Obsahuje tlačidlá pre odhlásenie a vytvorenie novej prezentácie. Nachádza sa tu zoznam vytvorených prezentácií a tlačidlá pre ich spravovanie.

Po kliknutí na hociktorý prvok zoznamu sa zobrazí detail prezentácie. Cieľom bolo vytvoriť panel, kde sa tvorca vie dozvedieť všetko o daných verziách prezentácie bez toho, aby musel otvoriť editor. Vie si tu zvoliť danú verziu a má náhľad k jej stránkam, ktoré si môže voľne preklikať.



Obr. 4.5: Domovská stránka



Obr. 4.6: Panel detailu prezentácie

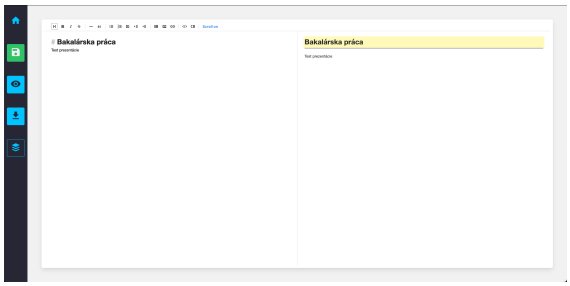
Editor

Hlavným cieľom pri návrhu editora bolo vytvorenie stránky s čo najväčším priestorom pre písanie obsahu a náhľadom sformátovanej stránky (viď. obrázok 4.7). Avšak dôležité bolo správne umiestnenie nástrojov. Nástroje a ovládacie prvky by mali byť jednoducho prístupné pre tvorcu prezentácie a nemali by byť príliš schované. Pre dosiahnutie veľkého priestoru pre obsah s jednoducho prístupnými ovládacími prvkami sa navrhol vysúvací bočný panel.

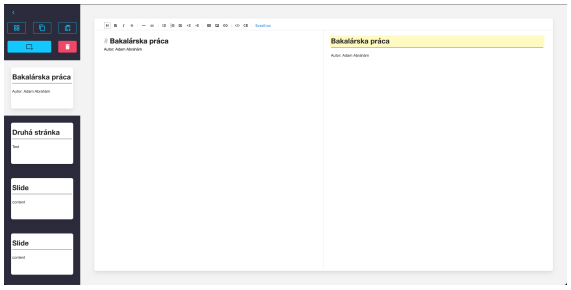
Bočný panel v zavretom stave zaberá minimum priestoru na stránke a obsahuje hlavné tlačidlá pre spravovanie prezentácie. Zobrazením stránok prezentácie sa bočný panel rozšíri, ako je vidieť na obrázku 4.8. V rozšírenom režime má tvorca prístup k zoznamu stránok, ktorý si môže zobraziť aj cez mriežkový pohľad pre jednoduchšiu orientáciu. Nad zoznamom sú dostupné tlačidlá pre prácu so stránkami.

Stránka prezentácie

Do prezentačného režimu sa užívateľ môže dostať cez domovskú stránku, ale aj cez editor. Na stránke je znázornený obsah prezentácie na celej obrazovke. Prezentácia je v tomto režime interaktívna a umožňuje prepínanie strán cez šípky klávesnice.



Obr. 4.7: Editor



Obr. 4.8: Rozšírený bočný panel

4.4 Návrh databázy

V tejto sekcii sa nachádza popis zvolených produktov pre databázu a popis uchovávanie údajov. Databáza obsahuje dve kolekcie. Nižšie sa nachádza popis oboch kolekcií a ich využitie v aplikácii. Pre jednoduchšie pochopenie, grafické znázornenie návrhu databázy je vidieť na obrázku 4.9.

MongoDB Atlas

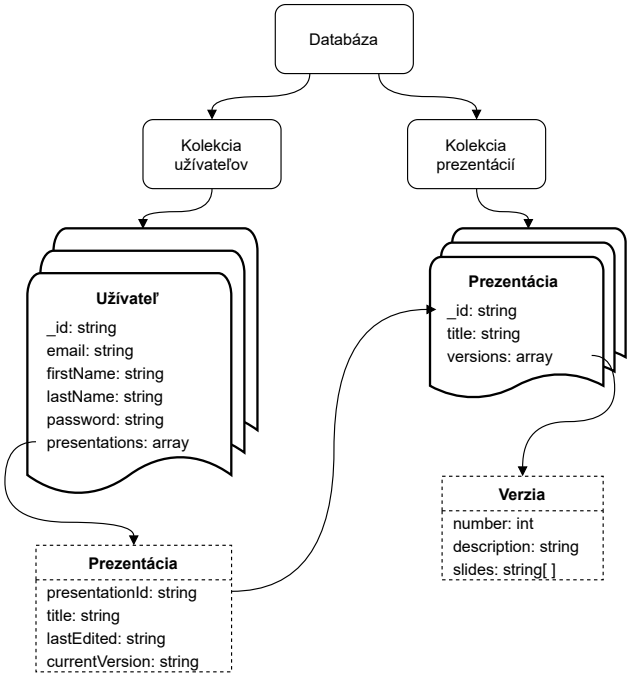
Na spravovanie a hosťovanie údajov sa používa cloudová databáza MongoDB Atlas. Pre použitie aplikácie vystačila bezplatná verzia hosťovania cez Amazon Web Services. Atlas ponúka webovú aplikáciu pre spravovanie, analýzu a zobrazenie štatistík databázy. Webová aplikácia je dostatočne zabezpečená, pre prístup k databáze je nutné pridať IP adresu do zoznamu povolených adries.

Kolekcia užívateľov

Kolekcia užívateľov uchováva dokumenty s informáciami o užívateľoch aplikácie. Nový dokument sa vytvorí pri registrácii nového užívateľa a obsahuje jeho osobné údaje, ako e-mail, krstné meno, priezvisko a heslo. Dokument naďalej obsahuje unikátny identifikátor a pole užívateľom vytvorených prezentácií. Objekty poľa uchovávajú stručné informácie o prezentácii, jej unikátny identifikátor, cez ktorý sa pristupuje ku všetkým informáciám prezentácie, dátum poslednej úpravy, názov a číslo aktuálnej verzie. K tejto kolekcii sa pristupuje po úspešnom prihlásení do aplikácie. Načítajú sa z nej údaje pre prihláseného užívateľa a informácie pre zoznam prezentácií na domovskej stránke.

Kolekcia prezentácií

Kolekcia prezentácií uchováva dokumenty s informáciami o vytvorených prezentáciách v aplikácii a o ich verziách. Dokument obsahuje unikátny identifikátor prezentácie, ktorý sa nachádza aj v zozname prezentácií užívateľa, názov prezentácie a pole verzií. Verzie v poli sú usporiadané vzostupne, od najstaršej k najnovšej. Jednotlivé objekty v tomto poli obsahujú číslo verzie, jej popis a pole obsahujúce stránky verzie. Stránky sú uchovávané v poli ako reťazce, od prvej k poslednej. Nový dokument v kolekcii sa vytvorí po uložení novej prezentácie. Nová verzia sa vytvorí v poli, keď užívateľ uloží prezentáciu ako novú verziu. Zvýšenie hodnoty čísla verzie má na starosti backend. Ku kolekcii sa pristupuje na troch miestach: na domovskej stránke pri otvorení panela detailu prezentácie, v editore pri upravovaní prezentácie a pri načítaní prezentácie v prezentačnom móde.



Obr. 4.9: Znázornenie návrhu databáze.

4.5 Návrh backendu

Serverová časť aplikácie prepojuje databázu s prezentačnou vrstvou. Obecne jej úlohou je sprostredkovanie komunikácie medzi nimi. Posiela HTTP odpovede na žiadosti klienta.

V tejto aplikácii má server za úlohu vytváranie, autentifikáciu, autorizáciu užívateľa, taktiež vytváranie, odstraňovanie, modifikovanie prezentácií a jeho verzií.

Návrh serverová časť sa skladá z troch hlavných vrstiev, ktoré medzi sebou komunikujú. Najnižšia je dátová vrstva, ktorá má za úlohu vytváranie mongoose schém databázy a mapovanie modelov a entít. Najvyššia vrstva je vrstva API, ktorá definuje koncové body a pristupuje k metódam repozitára. Tieto vrstvy sú prepojené vrstvou repozitára.

Vzor repozitára

V aplikácii je implementovaný vzor repozitára. Tento vzor pomáha v odstránení redundantného kódu. Zapúzdruje základné `CRUD` metódy nad databázou. V aplikácii sa nachádzajú dva repozitáre, repozitár užívateľa a prezentácie. K metódam repozitára sa pristupuje z API vrstvy.

```
repository
├── presentationRepository
└── userRepository
```

Repozitár užívateľa obsahuje následujúce metódy:

- `getUserById` - vyhľadávanie užívateľa podľa unikátneho identifikátora
- `getUserByEmail` - vyhľadávanie užívateľa podľa emailu
- `getUserToken` - získanie autentikačného tokenu užívateľa
- `saveNewUser` - uloženie nového užívateľa
- `newPresentationSummary` - vytvorenie nového stručného popisu prezentácie
- `updatePresentationSummary` - aktualizácia stručného popisu prezentácie
- `updatePresentationSummaryCurrentVersion` - aktualizácia aktívnej verzie prezentácie
- `deletePresentationSummary` - odstránenie stručného popisu prezentácie

V repozitári prezentácie sa nachádzajú metódy:

- `getPresentation` - získanie prezentácie podľa unikátneho identifikátora
- `newPresentation` - vytvorenie novej prezentácie
- `updatePresentation` - aktualizácia prezentácie
- `deletePresentation` - odstránenie prezentácie
- `deleteVersion` - odstránenie verzie prezentácie

Vrstva API

Vrstva API má za úlohu spracovávať dotazy od klienta na konkrétny koncový bod. Jednotlivé dotazy môžu obsahovať parametre v URI, takzvané query parametre, alebo parametre v tele dotazu. Pri úspešnom spracovaní dotazu API vráti HTTP odpoveď klientovi so správnym stavovým kódom `2xx`, kde čísla namiesto `x` sú doplnené podľa typu dotazu. Odpoveď môže obsahovať aj telo s obsahom. V aplikácii sa posiela obsah vo formáte JSON. Vrstva naďalej kontroluje, či je dotaz správny, či je užívateľ prihlásený a má dostačujúce oprávnenie na vykonanie dotazu, alebo či požadované dáta existujú. Pri odchytení chyby API vráti chybovú hlášku so stavovým kódom `4xx`.

Tabuľka 4.1: Koncové body

GET	`/api/user`	Získanie užívateľa
GET	`/api/user/presentations`	Získanie prezentácií užívateľa
GET	`/api/presentation`	Získanie konkrétnej prezentácie
POST	`/api/auth/register`	Registrovanie užívateľa
POST	`/api/auth/login`	Prihlásenie užívateľa
POST	`/api/presentation`	Uloženie, alebo aktualizácia prezentácie
DELETE	`/api/presentation`	Odstránenie prezentácie
DELETE	`/api/presentation/version`	Odstránenie konkrétnej verzie prezentácie

Kapitola 5

Implementácia

Táto kapitola popisuje čitateľovi samotnú implementáciu aplikácie. Čitateľ sa oboznámi s použitými nástrojmi pri tvorení aplikácie a s postupom implementácie frontendovej a backendovej časti. Prvým bodom implementácie bolo vytvorenie zložky mono-repozitára zahrňujúci obe časti aplikácie.

5.1 Git a GitHub

Pred začatím písania zdrojového kódu sa najprv spojazdnil verzovací systém Git cez GitHub. Cieľom bolo verzovanie samotného zdrojového kódu aj dokumentácie. Pre jednoduchšiu orientáciu v repozitároch sa vytvorila GitHub organizácia, ktorá obsahuje monorepozitár pre zdrojový kód a repozitár pre dokumentáciu na jednom mieste. Monorepozitár je repozitár obsahujúci viacero projektov, v tomto prípade projekt klienta a servera. Organizácia aj repozitáre sú verejne dostupné pod odkazom:

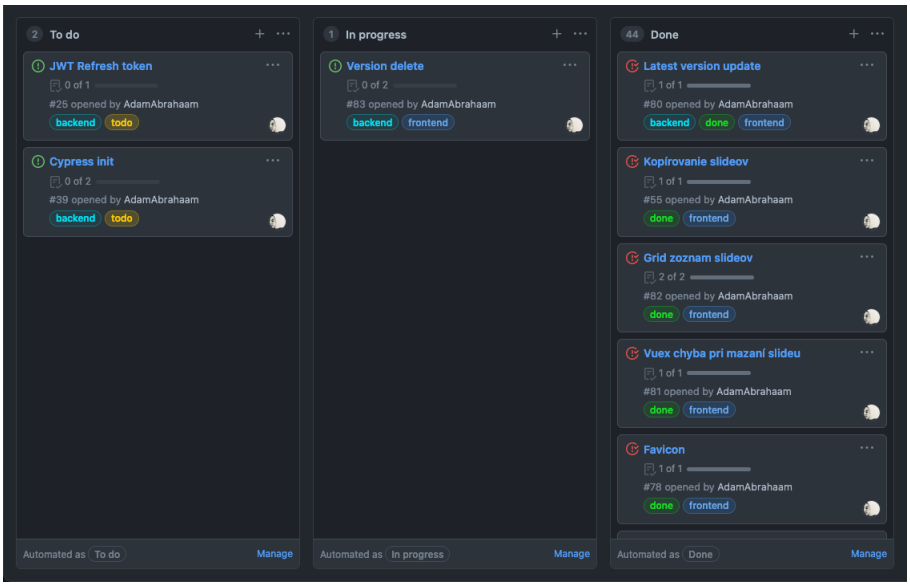
`https://github.com/AdamAbrahaamBC`

GitHub ponúka projektom organizačnú tabuľku(project board). Tabuľka pomáha v organizovaní úloh a poskytuje jednoduchší prehľad projektu. Môže mať ľubovolný počet kolónok, do ktorých sa prideľujú jednotlivé úlohy. V tomto projekte bolo postačujúce vytvorenie troch kolóniek. Kolónka `To Do` obsahuje ešte nezačaté úlohy, kolónka `In progress` úlohy, ktoré sa práve riešia a kolónka `Done` už dokončené a zatvorené úlohy. Pri vytvorení novej úlohy sa úloha automaticky priradí do kolónky `To Do`. Pri zvolení následujúcej úlohy musí byť manuálne premiestnená do kolónky `In Progress` a pri jej zatvorení sa automaticky premiestni do kolónky `Done`.

K jednotlivým úlohám sú pridelené štítky. Štítky označujú, o aký typ úlohy sa jedná. Dostupné štítky projektu sú nasledujúce.

- **backend** - označuje úlohy týkajúce sa backendu
- **frontend** - označuje úlohy týkajúce sa frontendu
- **db** - označuje úlohy na databáze
- **bug** - označuje úlohy obsahujúce chybu v aplikácii
- **todo** - označuje zatiaľ nezačaté úlohy

- **done** - označuje dokončené úlohy



Obr. 5.1: Organizačná tabuľka projektu

5.2 Implementácia frontendu

Implementácia frontendu sa začala stiahnutím `Node.js`. Node.js obsahuje nástroj `npx` pre spúšťanie balíčkov. Pomocou npx sa vytvoril Nuxt.js projekt cez terminálový príkaz: `npx create-nuxt-app <názov-projektu>`.

Po zadaní príkazu sa spustí sprievodca inštalácie, kde sa zvolil:

- správca balíčkov `npm`
- podpora pre `TypeScript`
- `Buefy` framework pre užívateľské prostredie
- komunikačný modul `axios`
- analyzačný nástroj zdrojového kódu `ESLint`
- `jednostránkový` typ aplikácie(SPA)

Balík Composition API je potrebné nainštalovať cez npm pomocou príkazu `npm install @nuxtjs/composition-api -save` a treba ho pridať medzi Nuxtom používané moduly v súbore `nuxt.config.js`.

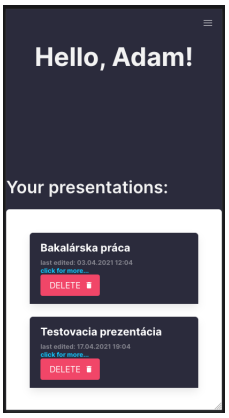
Súbor `nuxt.config.ts` treba hneď na začiatku zvýrazniť. Je to súbor obsahujúci konfigurácie aplikácie. Nastavujú sa v ňom vlastnosti hlavičky aplikácie, ako názov, meta značky a ikonka. Obsahuje nastavenia smerovania, TypeScriptu, autentifikačného modulu, globálnych CSS súborov a ostatných rozširovacích modulov.

Nuxt.js udáva jednoduchý štartovací bod pre vývoj aplikácie. Po inicializácii projektu skonštruuje štruktúru aplikácie, ktorá je doplnená adresármi `composable`, `cypress`, `models`. Štruktúru aplikácie dokopy tvoria nasledujúce adresáre:

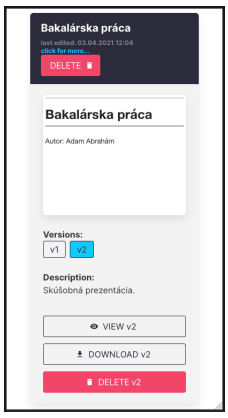
```
/client
├── /nuxt
├── /assets
├── /components
├── /composable
├── /cypress
├── /dist
├── /layouts
├── /models
├── /node_modules
├── /pages
├── /plugins
└── /store
```

Mobilná verzia

Aplikácie je určená hlavne pre webové prehliadače. Mobilná verzia je kompaktnejšia. Neobsahuje funkcionalitu vytvárania a upravovania prezentácií. Zobrazovanie a funkcie pre správu prezentácií sú avšak naďalej dostupné. Pohľad domovskej stránky a detailu prezentácie v mobilnej verzii je zobrazený na obrázku 5.2 a 5.3



Obr. 5.2: Mobilná verzia domovskej stránky



Obr. 5.3: Mobilná verzie detailu prezentácie

5.2.1 Autentifikácia užívateľov

Autentifikáciu užívateľov má na starosti autentifikačný nuxt modul. Modul sa nainštaluje cez npm, príkazom `npm install -save-exact @nuxtjs/auth-next` a pridá sa do nuxt konfiguračného súboru medzi moduly. Autentifikačný modul vyžaduje inštaláciu nuxt modulu `axios` pre HTTP komunikáciu.

Middleware je globálne nastavený v nuxt konfiguračnom súbore na každú jednu trasu aplikácie okrem stránky prihlasovania, registrovania a zobrazenia prezentácie v prezentačnom móde. Na týchto stránkach je middleware manuálne vypnutý a stránky sú dostupné

bez autentifikácie. Neprihlásený užívateľ pri navštívení stránky, ktorá autentifikáciu vyžaduje je automaticky presmerovaný na prihlasovaciu stránku.

Modul funguje na báze schém. Schémy definujú logiku autentifikácie. Projekt môže obsahovať viacero schém pre rôzne typy prihlasovania, napríklad cez Facebook, Google, atď. V aplikácii je implementovaná lokálna schéma na báze JWT tokenov. Jej konfigurácia sa nachádza v konfiguračnom súbore.

Modul poskytuje aplikačné rozhranie cez kľúčové slovo $auth, ktoré je globálne dostupné cez kontext aplikácie. Prihlasuje sa cez funkciu `$auth.loginWith('local', data: PRIHLASOVACIE_ÚDAJE )`. Modul pošle na prihlasovací koncový bod prihlasovacie údaje. Pri úspešnom prihlásení API vráti v HTTP odpovedi autentifikačný token užívateľa, ktorý sa uloží do koláčov(cookies) aplikácie a modul si automaticky vypýta údaje o užívateľovi podľa tokenu. Údaje užívateľa sú dostupné cez objekt `$auth.user`. Odhlasuje sa cez `$auth.logout()`.

5.2.2 Smerovanie a stránky

Smerovanie vo Vue.js sa rieši cez balík `vue-router`. Jednotlivé vlastnosti trias ako URI, komponenta pre vykreslenie a potomky je potrebné nakonfigurovať v konfiguračnom súbora.

Nuxt.js uľahčuje vývoj aj v tomto smere. Pomocou frameworku nie je potrebné konfigurovať trasy, Nuxt.js zoberie všetky zložky a súbory z adresára `pages` a skonštruuje smerovanie aplikácie automaticky. Každá zložka v adresári je samostatná trasa, ktorá obsahuje súbor `index.js` zahrňujúca komponentu stránky. Každá komponenta stránky je samostatná Vue.js komponenta so špeciálnymi atribútmi a funkciami. Zložky môžu obsahovať ďalšie vnorené zložky pre vnorené trasy. Zložky pomenované s podčiarkovníkom(_) na začiatku sú dynamické, ich hodnota sa mení. Dynamické stránky v aplikácii sa využívajú pre konkrétnu prezentáciu `/presentation/_id` a pre konkrétnu verziu prezentácie `/presentation/_id/_version`, kde sa tieto názvy menia podľa potreby. Obsah adresára:

```
/pages
├── /login
│   └── index.vue
├── /register
│   └── index.vue
├── /presentation
│   ├── /new
│   │   ├── /preview
│   │   │   └── index.vue
│   │   └── index.vue
│   └── /_id
│       └── /_version
│           ├── /preview
│           │   └── index.vue
│           └── index.vue
└── index.vue
```

5.2.3 Perzistentné úložisko

Uchovávanie jednotlivých stavov aplikácie je implementované cez modul `Vuex`. Aplikácia uchováva neuložené zmeny prezentácie a skopírovanú stránku. Pri opustení editora bez uložení zmien aplikácia upozorní užívateľa, že sa jeho zmeny môžu stratiť. Môže nastať situácia pri tvorbe prezentácie, keď sa z technických dôvodov aplikácia, alebo prehliadač zatvorí. V takomto prípade sa užívateľovi nové zmeny zachovajú a bude ich mať dostupné, keď sa vráti do editora danej prezentácie a verzie. Uchovávanie skopírovanej stránky sa využíva pri skopírovaní a vložení stránky prezentácie.

Štandardne sa uchované stavy vo Vuex po opustení aplikácie automaticky zmažú. Na perzistovanie úložiska sa využil balík `vuex-persistedstate`, pomocou ktorého sa stavy uložia do lokálneho úložiska prehliadača a sú dostupné aj pri znovu navštívení aplikácie.

Platí zásada, že stavy sa upravujú iba cez mutácie v úložisku. Mutácia sa volajú cez akcie, ktoré sú dostupné cez kľúčové slovo `store` v kontexte aplikácie. Implementácia samotného úložiska sa nachádza v adresári `store`. Zložka `presentation` obsahuje všetky potrebné súbory pre beh úložiska.

- súbor `index.ts` obsahuje uložené stavy prezentácie:
  - **presentation** posledná upravovaná prezentácia
  - **copiedSlide** skopírovaná stránka prezentácie
- súbor `mutations.ts` obsahuje mutácie stavov:
  - **SET_PRESENTATION** pridelenie hodnoty stavu prezentácie
  - **SET_COPIED_SLIDE** pridelenie hodnoty stavu skopírovanej stránky
- súbor `actions.ts` obsahuje metódy akcií nad mutáciami:
  - **SAVE_PRESENTATION** uloženie prezentácie do úložiska (volanie mutácie SET_PRESENTATION s hodnotou prezentácie)
  - **REMOVE_PRESENTATION** vymazanie prezentácie z úložiska (volanie mutácie SET_PRESENTATION s hodnotou `null`)
  - **COPY_SLIDE** uloženie skopírovanej stránky do uložiska (volanie mutácie SET_COPIED_SLIDE s hodnotou danej stránky)
- súbor `getters.ts` obsahuje metódy na vrátenie hodnôt stavov v úložisku:
  - **getPresentation** vráti hodnotu stavu prezentácie
  - **getCopiedSlide** vráti hodnotu stavu skopírovanej stránky

Pre zjednodušenie prístupu k úložisku a pre odstránenie biolerplate kódu sa vytvorila trieda `PresentationStore` v adresári `composable`. V jednotlivých komponentoch a na stránkach sa pristupuje k úložisku cez metódy tejto triedy. Trieda zapúzdruje vyššie popisované metódy akcií a metódy na vrátenie hodnôt stavov.

5.2.4 Znovupoužiteľná logika

Jednou z hlavných dôvodov použitia Composition API je jeho podpora pre jednoduché vyčlenenie znovupoužiteľnej logiky. Adresár `composable` obsahuje metódy a časti kódu využité na viacerých miestach v aplikácii. Príkladom je metóda pre získanie dát prezentácie v repozitári prezentácie, ktorá sa používa na domovskej stránke pri zobrazení detailu prezentácie, v editore pri úprave prezentácie a pri prezentovaní v prezentačnom móde.

5.2.5 Zobrazovanie a upravovanie Markdown obsahu

Zobrazovanie aj upravovanie Markdown obsahu sa rieši cez balík `toast-ui`. Toast-ui poskytuje komponentu `editor` pre upravovanie obsahu. Po vykreslení komponenty sa zobrazí textová oblasť, kde užívateľ môže zadať ľubovolný text. Nad editorom sa nachádza panel nástrojov obsahujúci jednotlivé Markdown nástroje. Na pravej strane editora sa nachádza náhľad sformátovaného Markdown obsahu. Pri každej strate zamerania(focus) editor vyvolá udalosť, na základe ktorej sa uložia zmeny obsahu do Vuex úložiska.

Komponenta `viewer` slúži na zobrazenie sformátovaného Markdown obsahu. Vykresľuje sa na domovskej stránke pri náhľade do jednotlivých verzií prezentácie a pri prezentácii obsahu v prezentačnom móde. Komponente je predaný obsah stránky vo formáte Markdown ako reťazec.

5.2.6 Extrahovanie statickej prezentácie

Užívateľ má možnosť exportovania prezentácie v statickom PDF formáte. Reveal.js poskytuje špeciálnu šablónu pre formát PDF. Po kliknutí na tlačidlo `download`, alebo na tlačidlo `export` na domovskej stránke, alebo v bočnom panely editora aplikácia presmeruje užívateľa do prezentačného módu, kde sa mu zobrazí dialóg pre správu tlače. V dialógu je potrebné zvoliť možnosť `Uložiť ako PDF`.

5.3 Implementácia backendu

Implementácia backendu začala inicializáciou projektu v mono-repozitári aplikácie cez terminálový príkaz `npm init -Y`. Značka `-Y` slúži pre automatické vytvorenie súboru `package.json` s predvolenými hodnotami. Po inicializácii projektu sa nainštaloval balík pre server `express` a balíky pre podporu TypeScriptu `typescript` a `ts-node`.

Celková implementácia a logika serverovej časti sa nachádza v adresári `server/src`. Začiatočným bodom pri spustení backendu je súbor `server.ts` nachádzajúci sa v hlavnom adresári. Súbor obsahuje triedu, pomocou ktorej sa spúšťa serverová časť. Pri vytvorení novej inštancie triedy sa nastavia všetky konfigurácie pre beh servera, ako port na odpočúvanie, explicitné povolenie JSON obsahu, podpora `.env` súborov a povolenie `CORS`. V triede sa pripája ku cloudovej MongoDB databáze cez pripojovací reťazec a inicializujú sa trasy aplikácie. Podrobnejšia implementácia trás sa nachádza v adresári `routes`.

5.3.1 Databáza

Databáza sa implementovala pomocou balíka `mongoose`. Mongoose poskytuje jednoduché pripojenie k MongoDB databáze, modelovanie objektov, vytváranie mongoose schém a dotazov. Prvým krokom bolo vytvorenie schém pre užívateľa a prezentáciu. Schémy sa mapujú

k jednotlivým dokumentom v databáze a definujú ich atribúty. Aby boli použiteľné, je potrebné ich namapovať na objekty rozhrania, takto vzniknú modely. Modely sú použité v repozitári, poskytujú jednotlivé funkcie na dotazovanie nad databázou. Schémy sa nachádzajú v adresári `database` a rozhrania modelov v adresári `models`.

Dotazovanie a logika nad databázou sa nachádza v adresári `repository`. V databáze sa nachádza kolekcia užívateľov a prezentácií. Pre obe kolekcie je vytvorený zvlášť súbor pre jednoduchšiu orientáciu. Repozitár užívateľov poskytuje funkcie pre získanie užívateľa podľa unikátneho identifikátora, získanie užívateľa podľa emailu, pridanie nového užívateľa do databázy, priradenie prezentácie k užívateľovi, odobranie prezentácie od užívateľa a upravenie krátkeho popisu prezentácie. Repozitár prezentácie poskytuje funkciu pre získanie detailu prezentácie podľa unikátneho identifikátora, uloženie prezentácie do databázy, odstránenie prezentácie z databázy, odstránenie konkrétnej verzie prezentácie a aktualizácia obsahu, alebo pridanie novej verzie prezentácie.

5.3.2 Aplikačné rozhranie

K aplikačnému rozhraniu sa pristupuje cez trasu `api`. Na backende sa pracuje s autentifikáciou, užívateľmi a prezentáciami. Pre každú jednu časť je definovaná zvlášť trasa. Základná definícia sa implementovala v súbore `server.ts`. Zadefinovanie koncových bodov a priradenie middlewaru k nim sa nachádza v adresári `routes`. Funkcie, ktoré sa zaoberajú spracovaním dotazov na jednotlivé koncové body od klienta sa nachádzajú v adresári `controllers`. Funkcie v týchto súboroch majú na starosť spracovanie jednotlivých dotazov, na základe požiadaviek volajú dotyčné funkcie v repozitároch a generujú odpoveď s patričným stavovým kódom.

5.3.3 Autentifikácia užívateľa

Autentifikácia užívateľa sa rieši cez JSON Web Žetóny(token). Pri prihlásení sa porovná užívateľom zadané heslo s heslom v databáze. Pri zhode sa podpíše token unikátnym identifikátorom užívateľa a tajným heslom, ktorý bol náhodne vygenerovaný špeciálnym generátorom a je uchovaný v `.env` súbore. Na podpísanie tokenu sa používa hašovací algoritmus `HMAC SHA256`. Token sa pošle spať HTTP odpoveďou. Autentifikačný modul v klientskej časti aplikácie token pridá do cookies a je naďalej posielaný pri každej komunikácii so serverom v hlavičke HTTP dotazu.

Autentifikácia užívateľa sa pomocou tohto tokenu kontroluje v middlewaru servera. Z HTTP hlavičky dotazu sa vytiahne token užívateľa, ktorý je uchovaný pod kľúčom `authorization`. V prípade, kedy kľúč neexistuje, alebo neobsahuje žiadnu hodnotu sa pošle odpoveď s chybovým stavovým kódom `401 Neautorizovaný`. Pri výskyte tokenu sa token verifikuje pomocou tajného hesla v súbore `.env`. Pri nezhode server odpovie chybovým stavovým kódom `403 Zakázané`. Pri úspešnom verifikovaní sa do HTTP dotazu pridá získaný unikátny identifikátor užívateľa a dotaz sa predá dotyčnej funkcie na jeho spracovanie.

Kapitola 6

Testovanie

V tejto kapitole sa nachádza popis a postup jednotlivých testov aplikácie, doplnený s obrázkami z testovacích prostredí. Pri testovaní sa použil nástroj Cypress a Postman. Práca s nimi je podrobnejšie popísaná nižšie.

6.1 Testovanie medzi dvomi stranami(End-to-End)

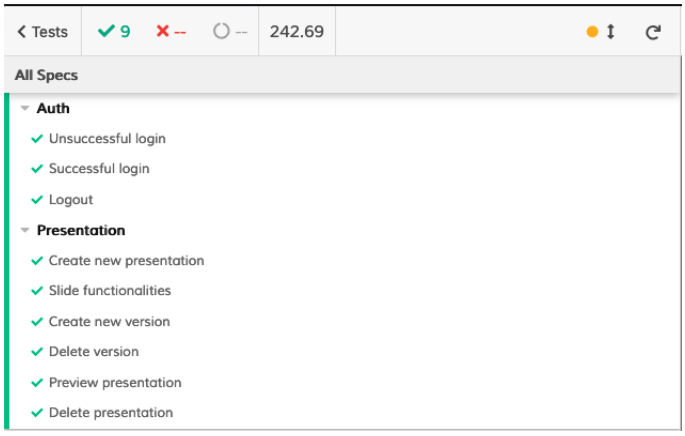
Testovanie medzi dvomi stranami, alebo End-to-End testovanie slúži na testovanie funkcionality celej aplikácie. Na rozdiel od jednotkových testov, ktoré testujú iba časť kódu, end-to-end testy prechádzajú celú aplikáciu z pohľadu užívateľa. Tieto testy sú dôležitou súčasťou aplikácie, umožňujú odchytenie chýb pri zmenách v zdrojovom kóde a zaručujú očakávané správanie od začiatku až do konca.

6.1.1 Cypress

Cypress je nástroj pre webové aplikácie, ktorý umožňuje vytváranie testov medzi dvomi stranami. Nástroj ponúka širokú škálu funkcionalít, pomocou ktorých sa dá otestovať každý jeden element webovej stránky. Cypress prechádza celú aplikáciu, simuluje správanie užívateľa. Umožňuje klikanie na tlačidlá, písanie do políčok, alebo aj presmerovanie na inú URL adresu. Poskytuje prehľadné užívateľské rozhranie, kde simuluje užívateľom vybraný prehliadač. Jednotlivé testy sa dajú spustiť naraz, ale aj zvlášť po častiach. Po vytvorení účtu a prihlásení, cypress uchováva históriu testov, ku ktorým sa dá hocikedy vrátiť. Nástroj umožňuje náhľad do testu v priebehu testovania, kde je vidieť každý jeden krok testu, s ktorými elementami sa pracovalo a aké akcie sa vykonali. K jednotlivým krokom sa v ktoromkoľvek okamihu počas testovania dá vrátiť.

Cypress sa inštaluje klasicky ako každý iný balík cez správcu balíčkov npm a spúšťa sa cez príkaz v termináli `npx cypress open`. Po prvom spustení sa v hlavnom adresári klienta vytvorí zložka `cypress`, s ktorou sa ďalej pracuje. Jednotlivé testy sa nachádzajú v zložke `cypress/integration`. End-to-End testy v aplikácii sú rozdelené na dve hlavné časti, `auth` a `presentation`. K jednotlivým elementom používaným v testoch sa pristupuje cez HTML data atribút. Tieto elementy sú v šablóne označené cez atribút `data-test`, napríklad: `<b-input data-test=ëmail"/>`. Pri ich potrebe sa element vyhľadá a zvolí pomocou cypress funkcie a vykoná sa potrebná akcia nad ním.

Po úspešnom priebehu, cypress označí test zelenou fajkou a pri narazení na chybu červeným krížom. Po dokončení testovanie sa zobrazí súhrn testovania a čas v sekundách trvania celého testu viď. obrázok 6.1.



Obr. 6.1: Súhrn dokončeného testovania

Autentifikačné testy

Autentifikačné testy sa nachádzajú v súbore `cypress/integration/auth.ts`. Tento súbor obsahuje testy:

- **Unsuccessful Login** - testuje sa zobrazenie chybovej hlášky pri neúspešnom prihlásení cez tlačidlo.
- **Successful Login** - testuje sa vyplnenie správnych prihlasovacích údajov a prihlásenie cez tlačidlo.
- **Logout** - testuje sa odhlásenie cez tlačidlo a presmerovanie na stránku prihlásenia.

Testy prezentácie

V súbore `cypress/integration/presentation.ts` sa nachádzajú všetky testy týkajúce sa prezentácií, od ich tvorby až k odstráneniu. Tento súbor obsahuje testy:

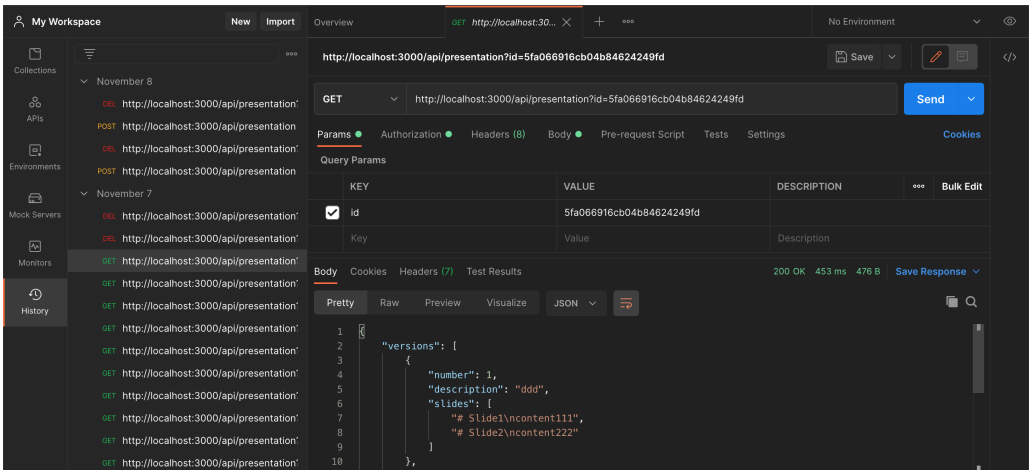
- **Create new presentation** - v tomto teste sa kontroluje, že daný užívateľ najprv nemá žiadne vytvorené prezentácie. Prezentácia sa vytvorí cez tlačidlo. `CREATE NEW` a otestuje sa, či URL adresa odpovedá adrese `/presentation/new` a či je editor viditeľný. Cez tlačidlo sa otvorí panel pre uloženie prezentácie a následne sa vyplnia potrebné údaje pre uloženie. Pomocou tlačidla sa vráti na domovskú stránku, kde sa skontroluje či zoznam obsahuje uloženú prezentáciu.
- **Slide functionalities** - v tomto teste sa kontrolujú všetko funkcionality nad stránkami prezentácie. Najprv sa otvorí prezentácia v editore cez tlačidlo na domovskej stránke. Cez tlačidlo sa otvorí bočný panel obsahujúci stránky prezentácie. Prezentácia v tomto okamihu obsahuje iba jednu stránku. Otestuje sa, že pri jednej stránke je tlačidlo odstránenia stránky nefunkčné. Nasleduje test kopírovania a vloženia stránky. Otestuje sa pridanie novej stránky cez tlačidlo a jeho odstránenie. V poslednom kroku sa kontroluje mriežkový pohľad stránok.
- **Create new version** - v tomto teste sa pracuje s verziami prezentácie. Znova sa prezentácia otvorí v editore, kde sa uloží pod iným názvom ako verzia číslo 2. Na domovskej stránke sa otestuje, že zoznam stále obsahuje iba jednu prezentáciu, ale s iným

názvom. Zobrazí sa detail prezentácie a skontroluje sa počet prezentácií. Otestujú sa rôzne popisy jednotlivých verzií. Otvorí sa verzia číslo 2 v editore a uloží sa pod iným názvom a rozličným popisom pod rovnakou verziou. Na domovskej stránke sa znova skontroluje počet prezentácií, počet verzií a či verzia číslo 2 obsahuje nové údaje.

- **Delete version** - v tomto teste sa rieši odstránenie verzie prezentácie. Mazanie sa uskutoční cez tlačidlo na domovskej stránke, aplikácia si vyžiada potvrdenie o mazaní. Po odstránení sa skontroluje počet verzií prezentácie.
- **Preview presentation** - v tomto teste sa testuje zobrazenie prezentácie v prezentačnom móde cez tlačidlo na domovskej stránke. Kontroluje sa či URL adresa obsahuje text `/preview`.
- **Delete presentation** - test kontroluje úspešné odstránenie celej prezentácie cez tlačidlo na domovskej stránke. Po odstránení sa testuje prázdny zoznam prezentácií.

6.2 Testovanie aplikačného rozhrania

Aplikačné rozhranie sa počas vývoja testovalo cez nástroj Postman. Postman je pomôcka, ktorá umožňuje zasielanie HTTP dotazov na koncové body a prijímanie odpovedí od nich. Nástroj poskytuje prehľadné užívateľské rozhranie, kde stačí zadať URL adresu koncového bodu. Užívateľské rozhranie obsahuje históriu poslaných dotazuv, ku ktorým sa dá kedykoľvek vrátiť. Postman taktiež umožňuj jednoduché pridávanie query parametrov, obsahu tela dotazu a upravovanie hlavičky dotazu. Odpoveď na dotaz je možné zobraziť vo viacerých formátoch. Prostredie nástroja je zobrazené na obrázku 6.2.



Obr. 6.2: Prostredie nástroja Postman

Kapitola 7

Záver

Cieľom tejto bakalárskej práce bolo vytvoriť webovú aplikáciu pre správu prezentácií s obsahom v značkovacom jazyku Markdown. Výsledná aplikáciu umožní užívateľom vytvárať, upravovať a odstraňovať jednotlivé prezentácie. Veľkou výhodou je možnosť verzovania prezentácií. Aplikácia poskytuje užívateľom možnosť uloženia úprav v prezentácii pod novou verziou a uchovanie starých verzií. Jednotlivé verzie sú voľne dostupné pre užívateľa, ktorý si ich môže kedykoľvek zobraziť, upraviť, stiahnuť, alebo odstrániť. Užívateľ má naďalej možnosť nahliadnuť do stránok verzie hneď na domovskej stránke, čo uľahčí voľbu tej správnej verzie pred jej zobrazením, alebo úpravou.

Editor obsahuje užitočné nástroje pre tvorbu Markdown obsahu s okamžitým náhľadom na sformátovanú stránku. Aplikácia umožňuje prehľadnú orientáciu medzi stránkami pomocou bočného panela a režimu mriežkového pohľadu. Postupnosť jednotlivých stránok sa dá jednoducho upraviť pomocou technológie ťahaj-a-pusti. Aplikácia naďalej implementuje podporu pre kopírovanie stránok medzi prezentáciami a ich verziami.

Serverová aj klientská časť aplikácie bola implementovaná v jazyku TypeScript. Voľbu TypeScriptu považujem za veľmi pozitívnu. Keďže jazyk som pomerne dobre ovládal už pred začatím implementácie projektu, jej použitie neprinieslo žiadne ťažkosti pri programovaní v nej, ale práve naopak. Jednotlivé časti aplikácie neboli hneď zdokumentované pri ich implementácii a po mesiacoch vrátenia k nim, zdrojový kód bol stále prehľadný a logika za ňou jednoducho pochopiteľná kvôli vlastnostiam TypeScriptu.

Klientská časť aplikácie bola implementovaná pomocou JavaScriptového frameworku Vue.js, s pomocou nadstavby Nuxt.js. Pri implementácii sa využila technológia Composition API, ktorá je dostupná vo Vue.js verzii 3. Po úspešnej implementácii projektu viem posúdiť, že tvorba komponentov a znovupoužiteľnej logiky cez Comosition API priniesli veľa výhod pri implementácii. Umožnila vyčlenenie znovupoužiteľnej logiky a jednotlivých častí zdrojového kódu, ktoré sa označujú ako boilerplate kódy a zaberajú veľa miesta, do zvlášť súborov. Výsledkom je prehľadnejšia orientácie v komponentoch aplikácie.

Pri implementácii sa použil verzovací systém Git a nástroj GitHub, ktorý pomohol v organizácii úloh projektu. Pri pridaní nových zmien do projektu sa testovala celková funkčnosť aplikácie pomocou nástroja Cypress, aby sa zaručilo očakávané správanie aplikácie.

Možným rozlíšením aplikácie by mohla byť funkcionalita porovnania dvoch verzií prezentácie, pričom by sa zvolili dve verzie a aplikácia by zobrazila rozdiely v ich obsahu. Porovnanie by naďalej zobrazilo vykonané akcie v novšej verzii, ako napríklad pridanie a odstránenie strán.

Literatúra

[1] Angular. *What is Angular?* [online]. Marec 2021 [cit. 2021-02-19]. Dostupné z: https://next.angular.io/guide/what-is-angular.

[2] Domantas, G. What is HTML? The Basics of Hypertext Markup Language Explained. [online]. 2019, [cit. 2021-02-18]. Dostupné z: https://www.hostinger.com/tutorials/what-is-html.

[3] Esplin, C. What is Firebase? [online]. Október 2016, [cit. 2021-02-20]. Dostupné z: https://howtofirebase.com/what-is-firebase-fcb8614ba442.

[4] Express. *Express, Fast, unopinionated, minimalist web framework for Node.js* [online]. [cit. 2021-02-25]. Dostupné z: https://expressjs.com/.

[5] Hunt, P. *Why did we build React?* [online]. Jún 2013 [cit. 2021-02-19]. Dostupné z: https://reactjs.org/blog/2013/06/05/why-react.html.

[6] InterWay. *SASS - CSS Preprocesor* [online]. [cit. 2021-02-18]. Dostupné z: https://www.interway.sk/blog/sass-css-preprocesor.html.

[7] Kvapil, J. 1. diel - Úvod do TypeScriptu. [online]. 2018, [cit. 2021-02-20]. Dostupné z: https://www.itnetwork.sk/javascript/typescript/uvod-do-typescriptu.

[8] MDN. *What is CSS?* [online]. [cit. 2021-02-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS.

[9] Microsoft Azure. *Databáze NoSQL – co je NoSQL?* [online]. [cit. 2021-02-17]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/nosql-database/.

[10] MongoDB. *Why and When to Use MongoDB* [online]. [cit. 2021-02-22]. Dostupné z: https://www.mongodb.com/why-use-mongodb.

[11] Mongoose. *Mongoose* [online]. [cit. 2021-02-22]. Dostupné z: https://mongoosejs.com/.

[12] Njenga, A. *10 Popular PHP frameworks to consider* [online]. Raygun, november 2018 [cit. 2021-05-06]. Dostupné z: https://raygun.com/blog/top-php-frameworks/.

[13] OpenJS Foundation. *About Node.js®* [online]. [cit. 2021-02-25]. Dostupné z: https://nodejs.org/en/about/.

[14] Python. *What is Python? Executive Summary* [online]. [cit. 2021-05-05]. Dostupné z: https://www.python.org/doc/essays/blurb/.

[15] Vue.js. *Getting Started* [online]. [cit. 2021-02-20]. Dostupné z: https://012.vuejs.org/guide/.

[16] WebSupport. *Čo je to databáza?* [online]. [cit. 2021-02-17]. Dostupné z: https://www.websupport.sk/faq/co-je-to-databaza.

Príloha A

Obsah pamäťového média

- `xabrah04.pdf` - PDF súbor tejto práce
- `xabrah04.zip` - zdrojové súbory tejto práce
- `src.zip` - zdrojové súbory aplikácie v monorepozitári
  - `server` - zdrojové súbory servera
  - `client` - zdrojové súbory klienta
  - `README.md` - spúšťací manuál