VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPRÁVU OBSAHU ZAMĚŘENÝ NA TÝMOVOU SPOLUPRÁCI

CONTENT MANAGEMENT SYSTEM FOCUSED ON TEAM COOPERATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DOMINIK JURIGA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Juriga Dominik**
Program: Informační technologie
Název: **Systém pro správu obsahu zaměřený na týmovou spolupráci**
Content Management System Focused on Team Cooperation
Kategorie: Web

Zadání:

1. Seznamte se s principy tvorby webových aplikací, dostupnými frameworky a současnými systémy pro správu obsahu webových stránek (CMS).
2. Analyzujte požadavky na CMS zahrnující správu obsahu, správu týmů, tvoření týmových plánů, poskytování informací o návštěvnících, blokový WYSIWYG editor a blog systém.
3. Navrhněte webový CMS splňující uvedené požadavky.
4. Navržený CMS implementujte a otestujte jeho funkčnost.
5. Zhodnoťte dosažené výsledky a diskutujte další možné pokračování této práce.

Literatura:

- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015. ISBN: 978-80-251-4573-9.
- Welling, L., Thomson, L.: PHP a MySQL: Kompletní průvodce vývojáře. CPress, 2017.
- Barker, D.: Web Content Management: Systems, Features, and Best Practices, 1st Edition. O'Reilly Media, 2016. ISBN: 978-14-919-0812-9.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

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

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 24. října 2019

Abstrakt

Táto bakalárska práca sa venuje návrhu, implementácii a testovaniu webovej aplikácie Contentu, ktorá spája funkcionalitu systémov pre správu obsahu so systémami pre správu tímu. Návrh tejto služby sa odvíja od analýzy požiadaviek a existujúcich riešení. Táto aplikácia je implementovaná pomocou technologického balíčka MERN, ktorý sa skladá z technológií MongoDB, Express, React a Node.js. Výsledkom práce je plne funkčná a otestovaná internetová služba, ktorej cieľom je spríjemnenie a zefektívnenie práce tvorcov obsahu.

Abstract

This bachelor thesis describes the design, development and testing of Contentu web application. This application bundles features of a content management system with features of a team management system. The design of this service is based on the requirement analysis and the analysis of existing solutions. This application uses the MERN technology stack, which consists of MongoDB, Express, React and Node.js. This thesis resulted in a fully functional and tested web service, which aims to simplify and streamline the workflow of content creators.

Kľúčové slová

webová aplikácia, správa obsahu, správa tímu, MongoDB, Express, React, Node.js, GraphQL, API, JavaScript, SPA

Keywords

web application, content management, team management, MongoDB, Express, React, Node.js, GraphQL, API, JavaScript, SPA

Citácia

JURIGA, Dominik. *Systém pro správu obsahu zaměřený na týmovou spolupráci.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Systém pro správu obsahu zaměřený na týmovou spolupráci

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vladimíra Bartíka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Dominik Juriga
28. mája 2020

Poďakovanie

Chcel by som sa poďakovať svojim rodičom za umožnenie tohto štúdia a za ich nekončiacu podporu. Taktiež ďakujem vedúcemu práce Ing. Vladimírovi Bartíkovi, Ph.D., ktorý mi poskytol cenné rady počas jej tvorby. V neposlednom rade by som chcel vyjadriť vďaku svojej partnerke a svojim priateľom, na ktorých som sa mohol vždy spoľahnúť.

Obsah

Kapitola 1

Úvod

Aktuálne využíva internet viac ako polovica ľudí na svete. Hlavnou motiváciou pre vznik internetu bolo zjednodušenie zdieľania informácií medzi viacerými zariadeniami a ich používateľmi. Jedným z najjednoduchších spôsobov, akým sa dajú informácie zdieľať celému svetu, je založenie vlastnej webovej stránky. Ku koncu 90. rokov minulého storočia boli vyvinuté systémy pre správu obsahu, ďalej označované *CMS (Content Management System)*, ktoré eliminovali potrebu upravovania kódu za účelom správy obsahu. Na trhu je dostupné veľké množstvo CMS a systémov na správu tímu. Nie je však žiadny systém, ktorý by tieto dva úkony spojil. Niektoré bežne dostupné CMS umožňujú integráciu so službami na správu tímu, často však býva potrebné manuálne importovať dáta medzi týmito systémami.

Tvorba webovej služby je komplexná úloha, ktorá zahŕňa dôslednú analýzu štandardov, princípov a technológií webu. Výber technológií môže ovplyvniť smerovanie, ale aj úspešnosť celého projektu. Preto je nutné zvoliť knižnice a aplikačné rámce, ktoré spĺňajú požiadavky projektu, ale aj jeho vývojárov. Text tejto práce uvedie čitateľa do sveta webu a jeho technológií. Vysvetlí pojmy a princípy, ktoré sú pri implementácii webovej služby nevyhnutné.

Kapitola 2 stručne popisuje vznik World Wide Webu a motiváciu za jeho vznikom. Taktiež predstavuje základné technológie, ktoré umožnili rozšírenie webu.

V kapitole 3 je prehľad dostupných technológií, ktoré sú zamerané na vývoj webových aplikácií.

Kapitola 4 sa venuje analýze dostupných riešení. Poskytuje porovnanie troch systémov pre správu obsahu a taktiež troch systémov pre správu tímu.

Kapitola 5 popisuje požiadavky, technológie a dizajn implementovaného systému. Taktiež popisuje návrh systému, jeho architektúru a dátovú štruktúru.

V kapitole 6 je popísaná implementácia systému, použité balíčky a nasadenie systému do produkčného prostredia.

V kapitole 7 je popísaná metodika testovania aplikácie, ktorá je neodlúčiteľnou súčasťou vývoja softvéru. Čitateľa uvedie do problematiky testovania webových aplikácií zo strany vývojára, ale aj používateľského testovania počas celosvetovej pandémie.

Záverečná kapitola 8 analyzuje priebeh projektu a hodnotí dosiahnuté výsledky. Nakoniec predstaví plány pre budúci vývoj tohto projektu.

Finálna verzia systému je dostupná na webovej adrese www.contentu.tech.

Kapitola 2

Web a jeho história

Táto kapitola ponúkne stručný náhľad na vznik World Wide Webu, jeho pôvodné technológie a ciele.

2.1 Vznik webu

Prvý návrh vytvoril britský vedec Tim Berners-Lee vo svojej práci *Information Management: a Proposal*[1] v roku 1989. Táto práca popisuje *World Wide Web* (často nazývaný WWW alebo W3), teda sieť, v ktorej sa informácie zdieľajú pomocou hypertextových dokumentov, ktoré môžu byť zobrazené pomocou prehliadačov. Počas tejto doby pracoval ako softvérový vývojár v Európskej organizácii pre jadrový výskum (CERN). Častým problémom bolo, že zamestnanci potrebovali prístup k informáciám, ktoré boli uložené iba v pamäti počítačov. Tieto počítače naviac často používali odlišný softvér, čo spôsobovalo komplikácie. Berners-Lee vedel, že v tej dobe už boli milióny počítačov prepojených pomocou siete internet. Navrhol teda spôsob zdieľania dát, na ktorý využil už existujúci Hyper Text. V niekoľkých nasledujúcich mesiacoch vytvoril tri technológie, ktoré sú dodnes základnými stavebnými kameňmi webu:

- Uniform Resource Identifier (URI)
- Hyper Text Transfer Protocol (HTTP)
- Hyper Text Markup Language (HTML)

Do konca roku 1990 vytvoril prvý webový server a prehliadač.V nasledujúcom roku bol web sprístupnený pre širokú verejnosť.[21]

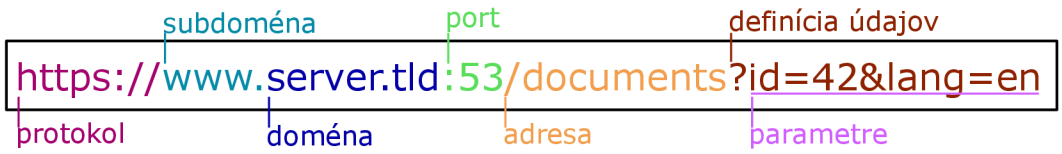
2.2 URI – Uniform Resource Identifier

URI je reťazec znakov, ktorý v kontexte počítačových sietí jednoznačne označuje určitý prostriedok. Na zaistenie jednoznačnosti využíva URI sadu syntaktických pravidiel. Samotný URI však poskytuje iba adresu, neimplikuje ani negarantuje prístup k danému prostriedku.[3] Na tento účel vznikol URL.

[1] www.cds.cern.ch/record/369245/files/dd-89-001.pdf

URL – Uniform Resource Locator

Pojmom URL sa označuje podmnožina URI, ktorá okrem jednoznačného označenia prostriedkov poskytuje aj spôsob prístupu pomocou špecifikovania prístupového mechanizmu. V kontexte počítačových sietí to znamená minimálne špecifikovanie prístupového protokolu (http, https, ftp...). URL sa skladá z viacerých častí, ktoré podliehajú syntaktickým pravidlám. Začína sa povinným označením protokolu, ktorý sa v bežne dostupných prehliadačoch automaticky dopĺňa na http alebo https. Za názvom protokolu nasleduje dvojbodka a dve lomky, za ktorými sa očakáva voliteľná subdoména a povinná doména. Nasleduje povinný argument port, ktorý sa značí dvojbodkou, za ktorou nasleduje číslo. Port sa opäť v prehliadačoch typicky dopĺňa automaticky a nie je vyžadovaný od používateľa. Ďalej je voliteľný argument adresa, ktorý špecifikuje adresu dokumentu v adresárovej štruktúre servera. V prípade absencie tohto argumentu je implikovaná adresa koreňového adresára. Pri požiadavkách požadujúcich koreňový adresár sa typicky zasiela dokument *index.html*. Posledným voliteľným argumentom je definícia údajov, značená znakom `?`, za ktorým nasledujú už spomínané údaje. Tieto údaje nazývame taktiež parametre. Parametre sú vo formáte `kľúč=hodnota` a vzájomne oddelené znakom `&`. Slúžia na bližšiu špecifikáciu požadovaného dokumentu, napríklad jazykovú verziu. Príklad URL je dostupný na obrázku 2.1.



Obr. 2.1: Príklad URL s vyznačenými časťami

2.3 HTTP – Hyper Text Transfer Protocol

HTTP je protokol aplikačnej vrstvy, ktorý slúži k prenosu dát medzi serverom a klientom. Dnes je možné označiť HTTP, vrátane jeho zabezpečenej verzie HTTPS, za hlavný spôsob prepravy informácií na internete. Na prenos informácií využíva protokol TCP. HTTP typicky využíva port číslo 80, pričom šifrovaná verzia HTTPS typicky využíva port číslo 403. Jedná sa o bezstavový protokol, ktorý pracuje na základe modelu požiadavka/odpoveď. Dôsledkom bezstavovosti je aj fakt, že HTTP nemá spôsob rozpoznávania klientov. HTTP špecifikuje u požiadaviek niekoľko druhov metód, ktoré určujú, akú akciu a s akým dátovým záznamom (resource) má server vykonať. Dostupné metódy a ich využitia sú nasledovné:

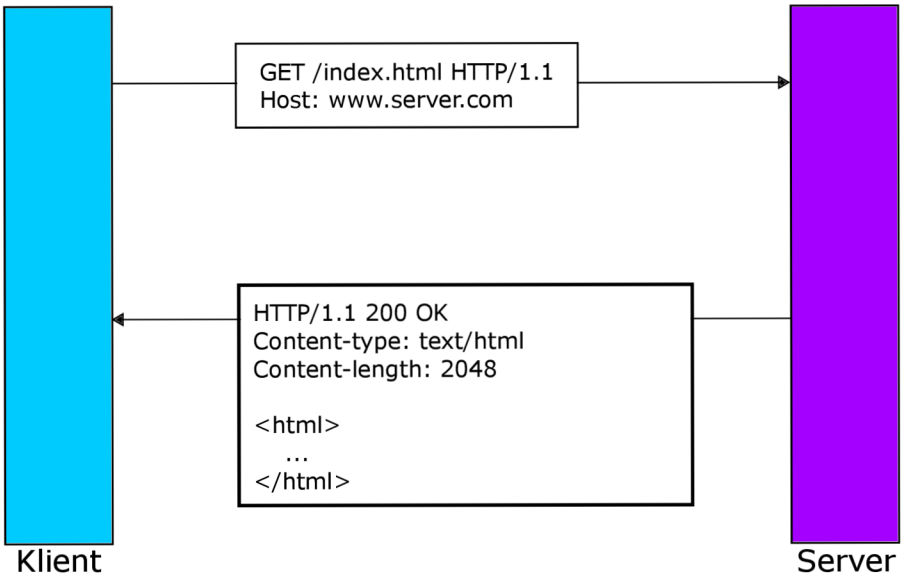
- GET – požaduje špecifický dokument, ktorý je definovaný v URL požiadavky
- HEAD – jedná sa o identickú požiadavku ako GET, no odpoveď nemá telo, iba metadáta v hlavičke
- POST – odoslanie dát na server, typicky spôsobuje zmenu stavu servera vytvorením nového dátového objektu
- PUT – odoslanie dát na server za účelom nahradenia celého existujúceho objektu dátovým obsahom požiadavky

- PATCH – odoslanie dát na server za účelom úpravy špecifických vlastností existujúceho objektu dátovým obsahom požiadavky
- DELETE – zmaže špecifický dokument
- CONNECT – spojenie klienta so serverom, typicky sa využíva pre SSL
- OPTIONS – požiadavka na deklarovanie dostupných metód, ktoré server podporuje
- TRACE – spätné odoslanie pôvodnej požiadavky klientovi na overenie spojenia

Odpovede sa skladajú z hlavičky a prípadne tela, pokiaľ klient vyžaduje nejaký dokument. Prvý riadok hlavičky odpovede je vyhradený pre stavový riadok, ktorý je vo formáte `verzia-HTTP odpoveďový-kód popis`. Odpoveďové kódy sa skladajú z troch číslic, pričom prvá z nich špecifikuje skupinu a ďalšie dve označujú špecifickú odpoveď. HTTP definuje 5 skupín, ktoré sú následovné:

- 100 – 199 – informačný charakter
- 200 – 299 – požiadavka skončila úspešne
- 300 – 399 – obsluha presmerovania, napríklad keď požadovaný dokument zmenil adresu
- 400 – 499 – klientova požiadavka obsahuje chybu
- 500 – 599 – pri obsluhe požiadavky nastala chyba na strane servera

Príklad HTTP komunikácie je dostupný na obrázku 2.2.



Obr. 2.2: Diagram HTTP komunikácie

Kapitola 3

Technológie vo webovom prostredí

Táto kapitola poskytne čitateľovi prehľad o technológiách, ktoré umožňujú prevádzku webových stránok a služieb. Taktiež ozrejmí pojmy, ktoré sú vo webovom prostredí zaužívané.

3.1 Webový server

Webový server slúži ako označenie pre hadvér, softvér, alebo obe. Z pohľadu hardvéru sa ako webový server označuje počítač, ktorý obsahuje softvér webového servera a prostriedky webových stránok, napríklad HTML dokumenty, obrázky a iné. Z pohľadu softvéru sa jedná o program, ktorý obsluhuje prichádzajúce požiadavky a na ich základe odosiela klientovi požadované prostriedky.

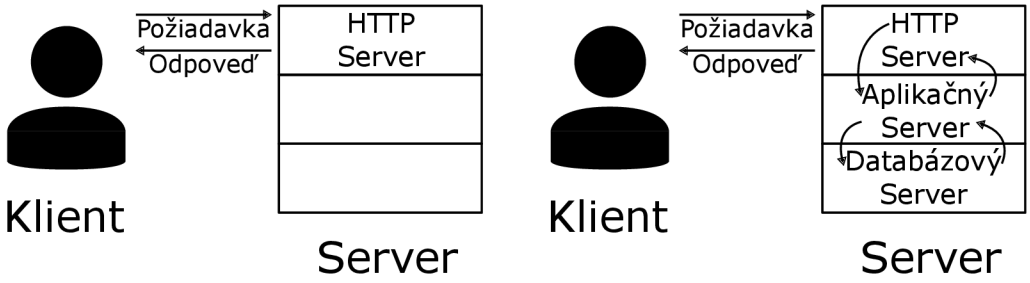
Po prijatí požiadavky webovým serverom (hardvér) je táto požiadavka delegovaná webovému serveru (softvér), ktorý sa postará o nájdenie príslušného dokumentu, prípadne jeho úpravu, a o jeho prenos späť ku klientovi. Webové servery sa delia na statické a dynamické. Postup pri spracovaní požiadavky je zobrazený na obrázku 3.1.

Statický webový server

Statický webový server produkuje statické webové stránky a typicky obsahuje iba dve časti. Prvou je HTTP server a druhou je úložisko súborov. Po prijatí požiadavky je nájdený príslušný dokument a ten je spätne odoslaný klientovi. Výhodou statického servera je jeho jednoduchosť a náročnosť na hardvér. Nevýhodou je, že informácie sú uložené priamo v dokumentoch, vďaka čomu je ich úprava náročnejšia ako u dynamických serverov. Využitie statického webového servera má zmysel pri stránkach, kde sa často nemenia informácie.

Dynamický webový server

Dynamický webový server produkuje dynamické webové stránky a obsahuje okrem statického webového servera aj aplikačný server a prípadne databázový server. Po prijatí požiadavky ju prevezme aplikačný server, ktorý sa postará o získanie príslušných informácií z databáze, vloží ich do požadovaného HTML dokumentu a výsledný dokument poskytne statickému webovému serveru, ktorý ho odošle klientovi.



(a) Statický webový server (b) Dynamický webový server

Obr. 3.1: Princíp získania dokumentu z webového servera

3.2 Webová stránka

Pojem webová stránka je možno chápať dvomi spôsobmi. Webová stránka, z anglického *web page*, je samostatný HTML dokument dostupný cez web, ktorý má svoj unikátny URI.[10] Webová stránka, z anglického *website*, je kolekcia navzájom prepojených webových stránok, ktoré zdieľajú doménové meno.[10]

3.3 Webové aplikácie

Podmnožinou webových stránok sú webové aplikácie. Webové aplikácie umožňujú rýchly a multiplatformný vývoj, pričom stačí jeden repozitár zdrojového kódu pre celú aplikáciu. Ako webovú aplikáciu označujeme takú webovú stránku, ktorá okrem iného poskytuje používateľom širokú interaktivitu a autentifikáciu. Jedná sa teda o dynamické webové stránky, ktoré pomocou interaktívneho používateľského rozhrania umožňujú meniť stav systému, alebo vykonávať nejakú komplexnú činnosť.

Bezpečnosť webových aplikácií

Vďaka globálnej povahe webu sú jeho služby častým cieľom útokov rôznych mierok. Bezpečnosť webových aplikácií je jedným z najdôležitejších vlastností takýchto služieb, pričom jej často nie je venovaná potrebná pozornosť. Medzi najčastejšie útoky patrí *SQL injection (SQi)*, pri ktorom útočník môže využiť neošetrené vstupy pri vykonávaní databázových dotazov, alebo *Cross Site Scripting (XSS)*, pri ktorom útočník využije bezpečnostné závady na vloženie svojich vlastných skriptov do zariadení ostatných používateľov. Týmto útokom sa typicky dá vyhnúť pomocou ošetrenia údajov získaných zo vstupných polí formulárov. Neexistuje žiadne pravidlo, ktoré by bolo aplikovateľné na všetky systémy, pričom ochrana pred útokmi sa odvíja od implementácie daného systému.

Menej typickým, no o to závažnejším, je *data breach* útok. Útočník sa pomocou využitia určitej bezpečnostnej chyby môže dostať k údajom, ktoré sú uložené v databáze. Obranu voči takýmto útokom je možné riešiť na viacerých úrovniach. Prvou úrovňou je zabezpečenie prístupu k databáze. Najčastejšie prichádza k týmto útokom vďaka nepozornosti alebo nevedomosti vývojárov, ktorí údaje k databázovému prístupu ponechajú v kóde aplikácie. Jednoduchým riešením tohto problému je uloženie údajov do tzv. premenného prostredia (anglicky environment variable), ktoré poskytuje operačný systém, a nie je možné ho zneužiť. Druhou úrovňou je zabezpečenie citlivých dát tak, aby nemohli byť škodlivo využité aj napriek ich kompromitácii. Citlivé údaje, ako napríklad heslá, by sa nikdy nemali v da-

tabáze ukladať v pôvodnej textovej forme. Existuje niekoľko spôsobov ukladania citlivých údajov, no najčastejším je využitie hašovania.

Zabezpečenie citlivých údajov pomocou hašovania

V kontexte počítačov sa na hašovanie údajov využívajú kryptografické hašovacie funkcie. Jedná sa o matematické algoritmy, ktoré transformujú vstupné dáta rôznej dĺžky na textový reťazec fixnej dĺžky, ktorý sa nazýva haš. Ideálna hašovacia funkcia spĺňa nasledujúce kritériá:

- Funkcia je jednosmerná a je nemožné nájsť k nej funkciu inverznú (jednosmernosť).
- Pre rovnaký vstup vždy vygeneruje rovnaký haš (deterministickosť).
- Je schopná za krátky časový úsek vygenerovať haš pre akýkoľvek vstup (rýchlosť).
- Nie je možné nájsť dva odlišné reťazce, pre ktoré by vznikol rovnaký haš (unikátnosť).
- Aj tá najmenšia zmena vo vstupnom reťazci by mala vygenerovať taký haš, ktorý s pôvodným hašom vôbec nesúvisí (lavínový efekt).

V rámci hašovania citlivých údajov sa typicky za účelom zvýšenia bezpečnosti porušuje kritérium deterministickosti. K tomu sa využíva tzv. kryptografická soľ. Jedná sa o reťazec niekoľkých bajtov, ktorý je pseudonáhodne generovaný a zreťazený s pôvodným vstupom. Takýto reťazec je následne použitý ako vstup hašovacej funkcie. Tým sa zaistí, že dvaja používatelia s rovnakým heslom nemajú v databáze uložený rovnaký haš. Vďaka kryptografickej soli sú údaje ochránené pred útokmi, ako je napríklad útok pomocou dúhovej tabuľky. Táto tabuľka obsahuje vopred vypočítané hodnoty umožňujúce zjednodušené prelomenie hašovaných údajov. Príklad zabezpečenia reťazca pomocou populárnych hašovacích algoritmoch je dostupný v tabuľke 3.1.

Pôvodný text	Toto je testovací reťazec znakov
MD5	9e8746d656a6db26019c354b2184b318
MD6-512	1c64218dda1ff8b2a70578366460a730fdb49054a62f263283d55 01ec322fc2bcce5ef43a26b67dd3731bb3f1034d2c07415d664f dc579155cf67510766dafbf
SHA-256	f5316c83e029bcfbc5ebbad71d44a3797c4634c64d81f6dc457a2 2140286a660
SHA-512	c838e76065a04a2cac2737d40d5cfc2e9a0003cb5f8b7821ad2f9 15b465058598f528587fa49e0cfa4e9095f4a7fd4fb80ed971766f 50085ab13671b456b0502
Bcrypt	2y12$gDBIJuz2VQfWKcabzYLqd.bnhp2JSsPPKRgxZEj/ T7p7BoR2Rh0Oq

Tabuľka 3.1: Príklad hašov vygenerovaných pomocou 5 hašovacích funkcií.

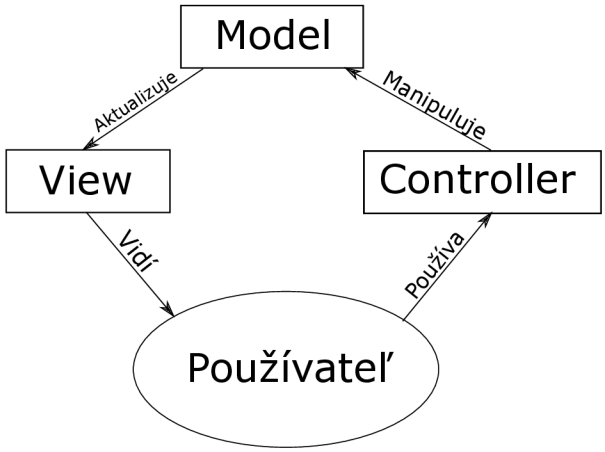
3.4 API - Application Programming Interface

API je rozhranie, ktoré obsahuje metódy alebo funkcie, a umožňuje dvom alebo viacerým programom medzi sebou komunikovať. Táto komunikácia môže byť jednosmerná alebo

obojsmerná a jej cieľom je prenos dát medzi týmito programami. V kontexte webových aplikácií sa využívajú koncové body API (ďalej označované ako endpoint), ktoré očakávajú HTTP požiadavky od klienta. Po prijatí sa požiadavka spracuje, zhromaždia sa potrebné dáta a pomocou HTTP odpovede sú odoslané späť ku klientovi.[6] API často plní úlohu dynamického webového servera.

3.5 MVC - Model View Controller

MVC je softvérová architektúra, ktorá rozdeľuje aplikáciu do troch častí, za účelom sprehľadnenia kódu. Oddeľuje od seba logiku, dáta a používateľské rozhranie, vďaka čomu si tímy pri vývoji nemusia zasahovať do práce. Taktiež zjednodušuje proces testovania, nakoľko je možné testovať tieto časti nezávisle na sebe. Diagram tejto architektúry je dostupný na obrázku 3.2.



Obr. 3.2: Diagram architektúry Model View Controller

Model

Model obsahuje dáta a logiku aplikácie, typicky sa jedná o komunikáciu s databázou alebo výpočty nad dátami. Model nezaujíma existencia iných častí systému, nakoľko ich pre svoju funkčnosť nutne nevyžaduje. Pri zmene stavu modelu spravidla upozorní na tieto zmeny svojich odoberateľov.[7]

View

View je vizuálna reprezentácia dát, ktorá je prezentovaná používateľovi, ktorý so systémom pracuje. Nemal by sa sám pokúšať získať prostriedky ani implementovať logiku aplikácie.[7]

Controller

Controller prijíma používateľské vstupy, transformuje ich na príkazy a deleguje ich do modelu.[7]

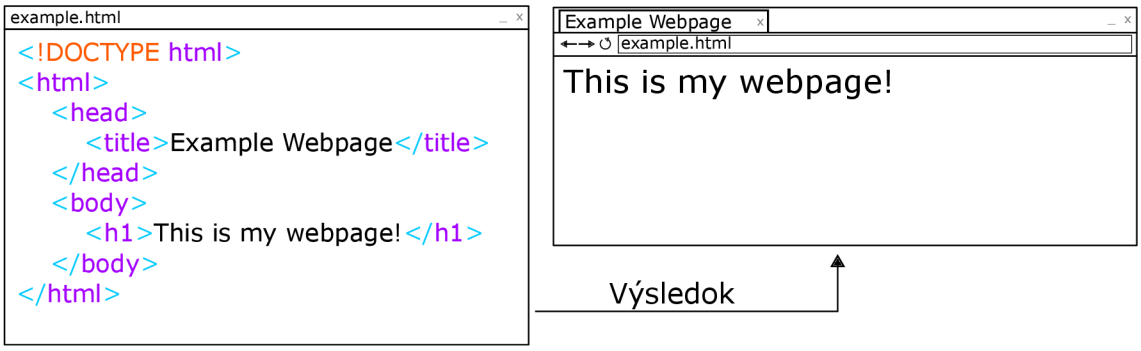
3.6 Technológie na vývoj frontend

Vývoj frontend je proces konvertovania dát systému na používateľsky prívetivé grafické rozhranie. Cieľom tohto procesu je poskytnutie rozhrania, ktoré umožní používateľom manipulovať dáta.

HTML - Hyper Text Markup Language

HTML je značkovací jazyk určujúci štruktúru dokumentov, ktoré majú byť zobrazené vo webovom prehliadači. Na oddelenie jednotlivých logických celkov dokumentu sú využívané

tzv. značky. Značky v HTML sú buď párové (napr. telo dokumentu), alebo samostatné (napr. obrázok). Existujú aj značky, ktoré sa dajú využiť v oboch podobách (napr. odkaz). [11] HTML dokumenty reprezentujú stromovú štruktúru, ktorá sa skladá zo značky *<!DOCTYPE html>* a značky *<html>*, ktorá obaľuje celý zvyšok dokumentu. Na nasledujúcej úrovni sa nachádza značka *<head>*, ktorá slúži na definíciu metadát (napr. názov stránky, cesta ku súborom určujúcim vzhľad alebo skriptom...) a značka *<body>*, ktorá obsahuje samotný obsah dokumentu. Po prijatí HTML súboru ho prehliadač transformuje na *DOM (Document Object Model)*. Príklad HTML dokumentu je na obrázku 3.3.



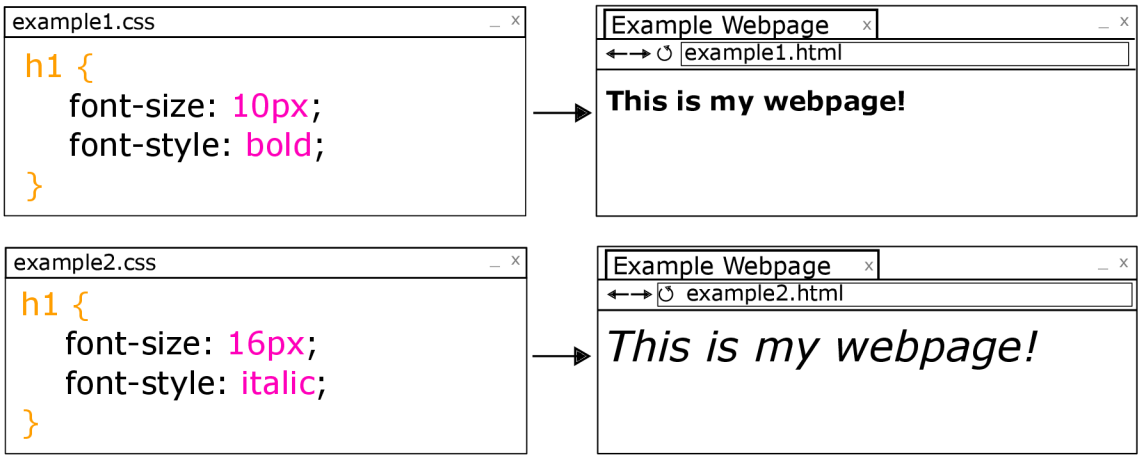
Obr. 3.3: Jednoduchý HTML dokument a jeho zobrazenie v prehliadači

Najnovšia verzia HTML5 prináša veľa zmien, ktoré sú zamerané najmä na čitateľnosť kódu, ale taktiež funkcionalitu.[12] Z pohľadu čitateľnosti kódu boli pridané nové sémantické značky. V starších verziách sa k deleniu dokumentu používala najmä značka * *, ktorá označuje časť dokumentu, na ktorú sa vzťahujú určité pravidlá. Nové značky sú napríklad *<header>* pre vrchnú časť stránky, *<nav>* pre navigáciu, *<section>* pre obsah, *<footer>* pre spodnú časť stránky. Z pohľadu funkcionality boli pridané značky *<video>* pre zobrazovanie videa, *<audio>* pre prehrávanie zvuku a *<canvas>* pre grafiku zobrazovanú pomocou JavaScriptu. Toto bolo v minulosti riešené pomocou rozšírení (napr. Flash), ktoré boli často pomalé a vytvárali bezpečnostné riziká.

CSS - Cascading Style Sheets

CSS je deklaratívny jazyk, ktorý určuje vzhľad dokumentov napísaných pomocou značkovacích jazykov. Medzi jeho hlavné úlohy patrí oddelenie definície vzhľadu od definície štruktúry dokumentu. Riadi sa špecifikáciou, ktorá je vydávaná konzorciom W3. Pri modernom vývoji webových stránok sa využívajú CSS preprocesory, napríklad SASS a LESS. Tie umožňujú dodatočnú funkcionalitu ako premenné, zanorovanie selektorov, funkcie... Vo výsledku sa prekladajú na CSS, no vývojárom uľahčujú prácu. Populárne sú taktiež predprogramované knižnice, medzi ktoré patrí napríklad *Bootstrap*, ktoré umožňujú rapídny vývoj responzívnych aplikácií kompatibilných s aktuálnymi prehliadačmi.

CSS dokument sa skladá z pravidiel, pričom každé pravidlo sa skladá zo selektora a deklarácie pravidla. Selektor označuje množinu prvkov dokumentu, na ktoré bude deklarácia pravidla aplikovaná. CSS špecifikuje dva špeciálne selektory. Prvým je `.`, ktorý vyberie prvky s danou triedou a `#`, ktorý vyberie prvky s daným identifikátorom. Deklarácia sa potom skladá z dvojíc `atribút: hodnota;`. Príklad jednoduchých CSS pravidiel je zobrazený na obrázku 3.4.



Obr. 3.4: Jednoduché CSS pravidlá a ich dopad na vzhľad dokumentu

JavaScript

JavaScript je objektovo orientovaný programovací jazyk určený na tvorbu dynamických dokumentov. Jeho najčastejšie využitie je na strane klienta, teda v prehliadači, kde sa využíva na tvorbu interakcií s používateľom a vytvorenie dynamického dojmu, alebo celého používateľského rozhrania pomocou frontend knižníc. Vďaka obrovskej podpore korporácií vyvíjajúcich prehliadače a úspechu technológií ako Node.js sa dnes využíva aj na tvorbu backend aplikácií a API. Spolu s HTML a CSS sa stal základnou technológiou pre vývoj webových stránok.

DOM - Document Object Model

Document Object Model je objektovo orientovaná reprezentácia HTML dokumentu. Umožňuje ďalším technológiám, ako napríklad JavaScript, aby manipulovali stránku pomocou svojho rozhrania. Úpravou obsahu dokumentu vzniká dojem dynamickej aplikácie. Podobne ako niektoré ďalšie webové technológie sa riadi pomocou špecifikácie, ktorá je vydávaná konzorciom W3. Vďaka tomu prišlo k zjednoteniu rozhraní dnešných prehliadačov, čo eliminuje potrebu *browser specific*[1] kódu.

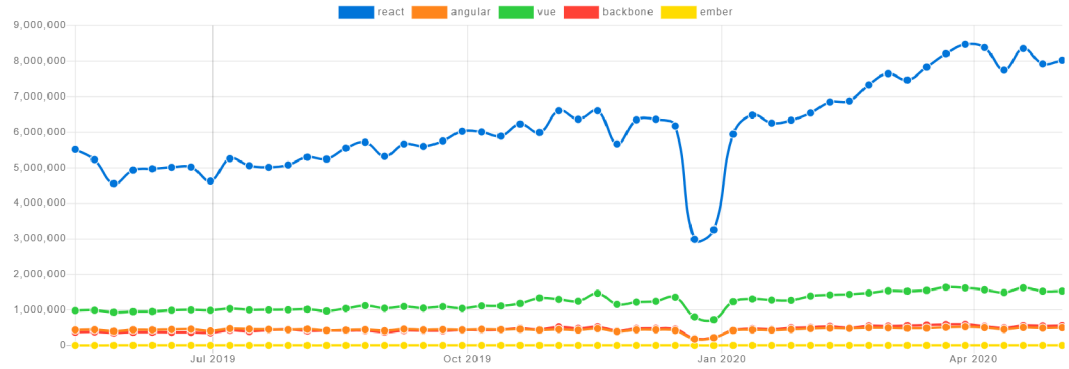
React

React je knižnica určená na vývoj jednostránkových aplikácií vyvinutá spoločnosťou Facebook. React reprezentuje pohľad (view) v MVC architektúre, slúži teda na vývoj používateľských rozhraní. Základným stavebným kameňom sú tzv. znovupoužiteľné komponenty, ktoré sa delia na triedne a funkcionálne, a majú svoj stav. Jedná sa o vykresľovanie na strane klienta. Server odosiela iba základný HTML súbor obsahujúci metadáta a prázdny uzol *root*, ktorý je po získaní dát na strane klienta týmto obsahom vyplnený. Hlavnou výhodou aplikácií vyvinutých pomocou knižnice React je podmienené prekreslovanie. To znamená, že jednotlivé komponenty sa prekreľujú iba na základe zmeny ich stavu. Pri prechode medzi jednotlivými stránkami aplikácie sa teda nemusí prekresľovať celé okno, ale iba komponenty, v ktorých sa menia dáta.[13]

[1] kód, ktorý zaisťuje funkcionalitu pre daný prehliadač, ktorý nerešpektuje špecifikácie konzorcia W3

React využíva na definovanie štruktúry komponentov technológiu *JSX*. Jedná sa o rozšírenú syntax jazyka JavaScript, ktorá je inšpirovaná jazykom HTML. Okrem jednoduchej definície elementov v prostredí React umožňuje taktiež ich ukladanie do premenných a ich následnú manipuláciu.[5]

Aj napriek tomu, že React reprezentuje iba jednu vrstvu MVC, je často radený medzi JavaScript frameworky, ktoré poskytujú všetky 3 vrstvy MVC. React je spomedzi týchto frameworkov najpopulárnejší. Porovnanie populárnosti moderných frameworkov je zobrazené na obrázku 3.5.



Obr. 3.5: Popularita JavaScript frameworkov podľa počtu stiahnutí za posledný rok[2]

Angular

Angular je framework na vývoj dynamických webových aplikácií. Tento framework je založený na komponentoch, ktoré môžu reprezentovať jednotlivé elementy MVC architektúry. Vďaka tomuto je možné v tomto prostredí vytvoriť celú webovú službu. Je to open-source framework založený na jazyku TypeScript, ktorý vznikol vďaka firme Google, ktorá sa postarala o kompletné prepísanie pôvodného frameworku AngularJS.

Angular poskytuje svoje vlastné rozhranie v príkazovom riadku, vďaka čomu je možné pomocou niekoľko málo príkazov vytvoriť, dokumentovať, otestovať a nasadiť projekt. Súčasťou Angularu je AOT prekladač[3], ktorý pri budovaní produkčných súborov konvertuje TypeScript a HTML kód na JavaScript, vďaka čomu sa tieto stránky načítajú používateľom rýchlejšie.

3.7 Technológie na vývoj backend

Vývoj backend typicky predstavuje proces tvorby softvéru, ktorý slúži ako aplikačný server v dynamickom webovom serveri. Tento softvér je určený na spracovanie požiadaviek, získavanie príslušných dát a ich odoslanie klientovi.

REST - REpresentational State Transfer

REST je architektúra, ktorá sa dnes bežne využíva pre vývoj webových služieb, najmä API. Systémy využívajúce túto architektúru sa označujú ako *RESTful*.[20] Takéto systémy vo

[2] www.npmtrends.com/react-vs-angular-vs-vue-vs-backbone-vs-ember
[3] Ahead Of Time - vopred vykonaný preklad zdrojových súborov

väčšine prípadov využívajú na komunikáciu protokol HTTP. Táto architektúra funguje na princípe endpointov. Systém definuje zoznam endpointov, ktoré sú identifikované pomocou URI (Uniform Resource Identifier) a ich dostupných HTTP metód. Typicky sa pre jeden druh záznamov definujú dva endpointy, prvý z nich umožňuje získať všetky záznamy toho typu, druhý z nich umožňuje získať špecifický záznam. Príkladom je napríklad zoznam používateľov, pomocou GET požiadavky na URI *http://example.com/users* môžeme získať zoznam používateľov a pomocou GET požiadavky na URI *http://example.com/user/User42* môžeme získať údaje o používateľovi s identifikátorom *User42*.

Svoju popularitu si získal najmä vďaka svojej jednoduchosti a všestrannosti. RESTful systémy vďaka svojej jednoduchosti však trpia dvomi zásadnými nedostatkami, *under-fetching* a *over-fetching*. Under-fetching je situácia, kedy klient potrebuje aj dáta, ktoré však endpoint neposkytuje, vo výsledku musí kontaktovať viacero endpointov, čím vzniká zbytočný počet požiadaviek na systém. Opakom tejto situácie je over-fetching, kedy endpoint odosiela viacero dát, než klient potrebuje. Toto nemusí byť vždy problematické, no systém môže zasielať veľké množstvo dát, čo môže na strane klienta spôsobiť spomalenie systému a zbytočne zahlcovať sieť. Jednoduchým riešením je vytvoriť špecifické endpointy na strane systému, no to zasa vedie ku zbytočnej komplikácii kódu. Taktiež býva problém s aktualizáciami týchto systémov, nakoľko môže dôjsť ku znefunkčneniu závislých aplikácií.

GraphQL

GraphQL je dotazovací jazyk vyvinutý spoločnosťou Facebook. Vznikol ako odpoveď na typické nedostatky RESTful systémov, najmä over-fetching alebo under-fetching. Namiesto množstva „hlúpych“ endpointov využíva GraphQL jeden „múdry“ endpoint. Tento endpoint umožňuje klientovi definovať dáta, ktoré v daný moment potrebuje pomocou požiadaviek (query) a mutácií (mutation). Tieto systémy môžu byť taktiež jednoducho aktualizované a môžu pridávať novú funkcionalitu bez toho, aby boli závislé systémy akokoľvek zasiahnuté.

Príklad jednoduchej požiadavky v jazyku GraphQL[4]:

```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

Odpoveď z GraphQL servera je prijatá vo formáte JSON:

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [{
          "name": "Luke Skywalker"
        }]
    }
  }
```

[4] https://graphql.org/learn/queries/

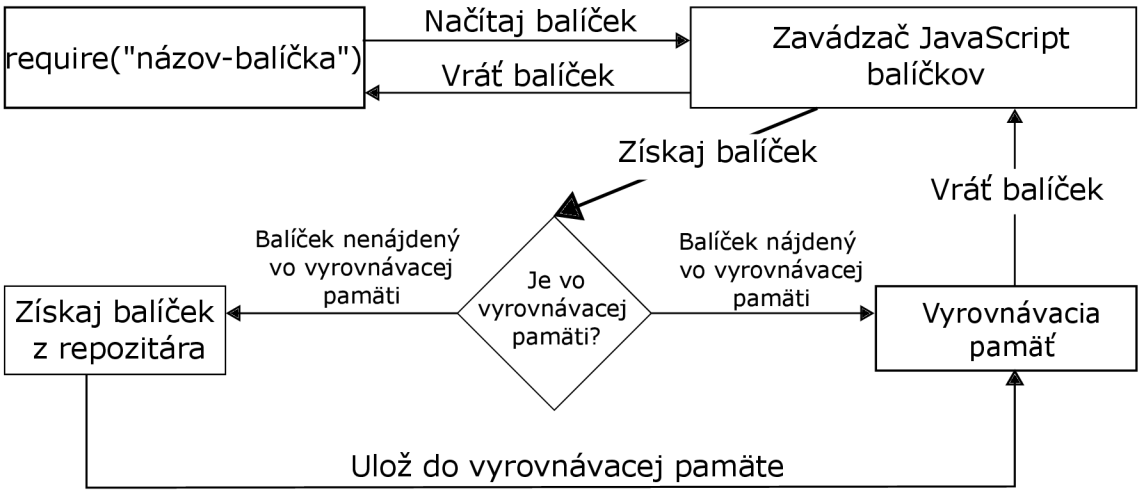
GraphQL definuje schémy pre jednotlivé modely systému. Schéma popisuje, ako vyzerá samotný model objektu systému, a aké požiadavky a mutácie je možné nad ním vykonávať. Každá schéma má tzv. resolver, ktorý sa stará o spracovanie konkrétnej požiadavky a odoslanie príslušných dát.[19]

PHP - Hypertext Preprocessor

PHP je imperatívny objektovo orientovaný jazyk, ktorý sa využíva najmä na tvorbu serverových aplikácií a API. Jeho prvá verzia vyšla v roku 1995 a až donedávna bol hlavnou technológiou pre vývoj webových aplikácií, vďaka čomu má obrovskú komunitu vývojárov. Aktuálne je PHP využívané v 78.8% všetkých webových stránok, no toto číslo pomaly klesá. [16] Obľúbený je najmä kvôli jeho jednoduchosti a rozšírenosti rôznych frameworkov, ako napríklad český framework Nette, Laravel alebo Symfony. Medzi často vytýkané chyby PHP patrí rýchlosť. s príchodom verzie PHP7 v roku 2015 však prišlo k zlepšeniu, a oproti predošlým verziám je v určitých situáciách niekoľkonásobne rýchlejšia. Na základe testovania firmy Kinsta, ktorá porovnávala rýchlosť spracovania požiadaviek rôznych verzií PHP medzi rôznymi CMS bolo zistené, že PHP7 je rýchlejšie v každom CMS, pričom maximálny rozdiel v prospech PHP7 dosiahol až 320%. [8]

Node.js

Node.js je behové prostredie (runtime), ktoré umožňuje spúšťat JavaScript programy mimo prostredia webového prehliadača. Node.js je postavené na výkonnom open-source JavaScript interprete *V8*, ktorý poskytuje firma Google. [14] Samostatne však Node.js veľa funkcionality neposkytuje, jeho skutočný potenciál sa skrýva v balíčkoch. Node.js obsahuje nástroj na správu balíčkov, ktorý sa nazýva *npm* (Node Package Manager). Balíčky sa inštalujú pomocou príkazu `npm install nazov-balíčka`, ktorý vytvorí novú položku v súbore *package.json*. Pre zdieľanie projektu stačí odoslať zdrojové kódy a tento súbor, následne stačí použiť príkaz `npm install`, ktorý doinštaluje všetky potrebné balíčky. Tieto balíčky sú uložené v zložke *node_modules*. To umožňuje jednoduchú kolaboráciu medzi jednotlivými používateľmi, alebo viacerými zariadeniami. Obrázok 3.6 zobrazuje postup načítania jednotlivých balíčkov do programu.



Obr. 3.6: Princíp načítania npm balíčkov do programu

JWT - JSON Web Token

JWT je elektronický kľúč (token), ktorý umožňuje bezpečný prenos dát medzi klientom a serverom. Každý token sa skladá z hlavičky (header), dát (payload) a podpisu (signature). JWT sa nesnaží skryť obsah dát, ale zaistiť ich autenticitu. K tomu slúži podpis, ktorý typicky kóduje hlavičku a dáta pomocou algoritmu HMAC-SHA256 privátnym kľúčom servera. Pred odoslaním sa pre zníženie nákladov na prenos každá časť tokena zakóduje pomocou kódovania base64. [2]

3.8 Databázové systémy

Hlavnou úlohou aplikácií a počítačových programov všeobecne, je uľahčenie alebo urýchlenie práce s informáciami. Tieto informácie môžu, ale nemusia byť dlhodobo skladované v úložisku počítača. V prípade webových aplikácií je vo väčšine prípadov najvhodnejším spôsobom ukladania dát využitie databázových systémov. Tie umožňujú *CRUD*[5] funkcionalitu nad dátami a ich dynamické filtrovanie na základe špecifikovaných požiadaviek. Databázové systémy sa delia na základe štruktúry, v ktorej dáta uchovávajú. Prvým typom sú relačné databázy (taktiež označované ako SQL databázy), pretože ich štruktúra vychádza z matematického relačného modelu. Druhým typom sú NoSQL databázy, ktoré využívajú rôzne štruktúry a často sa špecializujú na jednu nefunkčnú požiadavku (napríklad dostupnosť, rýchlosť. . .).

Databázy založené na relačnom modeli

Jedná sa o jeden z najstarších databázových modelov. Jeho autorom je Edgar Codd, ktorý bol nespokojný s doterajším stavom databázových systémov, a preto v roku 1970 vydal publikáciu s názvom *A Relational Model of Data for Large Shared Data Banks*[6], v ktorej popisuje systém, do ktorého sa môžu voľne vkladať, mazať alebo upravovať dáta, pričom zodpovednosť za vykonanie týchto operácií je prenechaná databázovému systému. [4] Dáta sú na logickej úrovni ukladané do tabuliek, pričom každý stĺpec určuje atribúty jednotlivých záznamov a riadok je jeden záznam, ktorý je označený unikátnym identifikátorom nazývaným primárny kľúč. [15]

O niekoľko rokov neskôr sa objavil štandardizovaný jazyk SQL (Structured Query Language), ktorý poskytuje príkazy na definíciu a manipuláciu dát, ale aj príkazy pre kontrolu dát, ktoré umožňujú udeľovanie prístupových práv alebo riadenie transakcií. Jednou z hlavných požiadaviek na relačné databázové systémy je súbor požiadaviek *ACID*[7], ktorý zaisťuje okamžitú integritu dát v databáze aj v prípade neočakávaných udalostí, ako napríklad pád systému alebo výpadok elektriny. Predstaviteľmi relačných databáz sú *MySQL*, *PostgreSQL*, *Oracle*.

NoSQL databázy

NoSQL databázy pôsobia ako protipól ku tradičným relačným databázam, nakoľko nie sú založené na relačnom modeli. Pojem ako taký sa zakorenil až v 21. storočí, no prvé databázy by mohli taktiež byť označené ako NoSQL. Názov NoSQL evokuje dojem, že tieto databázové systémy nepodporujú dotazovacie jazyky, no v skutočnosti je vykladaný

[5] z anglického Create, Read, Update, Delete - vytvor, čítaj, aktualizuj, zmaž
[6] www.seas.upenn.edu/ zives/03f/cis550/codd.pdf
[7] z anglického Atomicity, Consistency, Isolation, Durability - atomicita, konzistencia, izolácia, trvalosť

ako „Non Relational“ alebo „Not Only SQL“. To znamená, že tieto databázové systémy majú svoj vlastný dotazovací jazyk, prípadne sú využívané v tandeme s inými databázami v architektúre *polyglot persistence*[8]. NoSQL databázy nemusia mať dopredu definovanú štruktúru, a preto sa môže líšiť systém od systému. Hlavným spôsobom ukladania dát sú dokumenty, kde sa dáta ukladajú prevažne ako JSON objekty, prípadne BSON[9]. Ďalej môžu byť dáta ukladané vo forme grafov *(Neptune)*, kľúč-hodnota *(DynamoDB)*, vo vyrovnávacej pamäti (Redis)... Predstaviteľmi NoSQL databáz sú *MongoDB*, *Cassandra*, *Redis*.[1]

[8]architektúra, pri ktorej sú v jednom systéme pre rôzne dáta využité rôzne spôsoby ukladania dát

[9]Binary JSON, formát, do ktorého sú zakódované JSON objekty pomocou niektorého serializačného algoritmu (napr. Base64)

Kapitola 4

Analýza existujúcich riešení a ich architektúra

Táto kapitola predstaví čitateľovi existujúce riešenia systémov pre správu obsahu a správu tímu.

4.1 Informačné systémy

Informačné systémy sú v rovine softvéru podmnožinou webových aplikácií, ktoré poskytujú rozhranie na zber, spracovanie, ukladanie a zdieľanie informácií. Často sa však ako informačný systém označuje celok, ktorý je zložený z aplikácie, ľudí, ktorí ju používajú a procesov, ktoré títo ľudia v aplikácii vykonávajú. Existuje mnoho druhov informačných systémov, napríklad systémy na plánovanie hmotných a ľudských zdrojov, dodávacieho reťazca a iné.

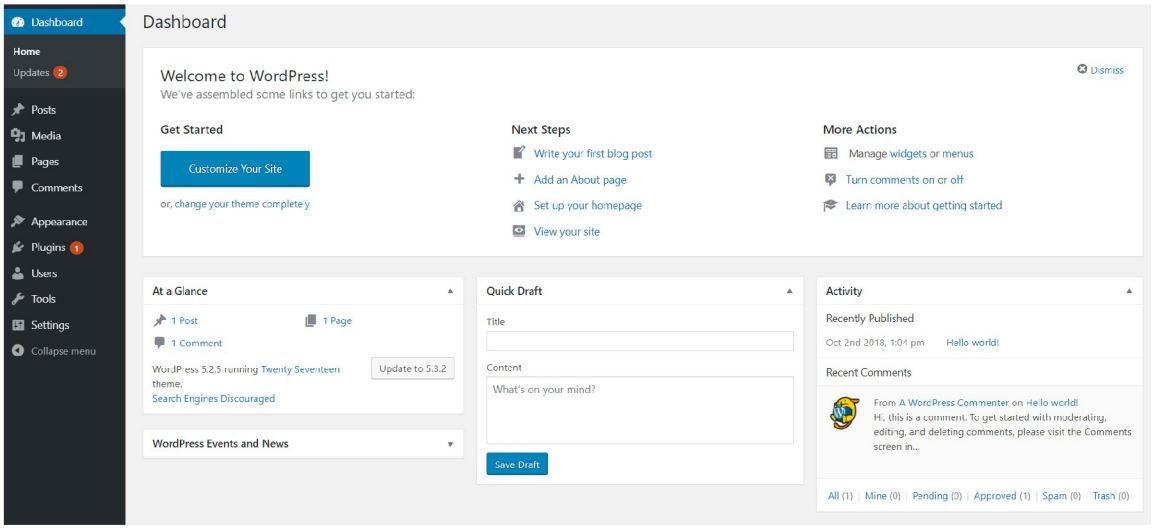
4.2 Existujúce systémy pre správu obsahu

V dnešnej dobe je možné vybrať si doslova zo stoviek CMS. Celému trhu však už niekoľko rokov dominuje WordPress. Z verejne dostupných údajov vyplýva, že viac ako 62% webových stránok, ktoré používajú CMS, využíva WordPress [17]. V nasledujúcom texte bude predstavených niekoľko z nich.

WordPress

WordPress[1] patrí medzi najstaršie a najpoužívanejšie CMS na trhu. V počiatkoch sa jednalo o systém na tvorbu a správu blogov, postupne sa však rozrástol, a dnes by sa ťažko dalo vymenovať všetky jeho využitia. Jedná sa o open-source systém, ktorý poskytuje možnosť využitia vlastných tém, a taktiež vlastných rozšírení (plugin), vďaka čomu je jeho funkcionalita skoro nekonečná. Toto však v spojení s popularitou prináša bezpečnostné riziká. Webové stránky spravované pomocou WordPressu bývajú často cieľom útokov, preto je potrebné často aktualizovať celý systém a všetky nainštalované rozšírenia. Podľa bezpečnostnej firmy Sucuri bolo v roku 2019 pri infikácii WordPress stránok iba 51% z nich aktualizovaných na poslednú verziu. [9]

[1] www.wordpress.com



Obr. 4.1: Používateľské rozhranie systému WordPress

NetlifyCMS

NetlifyCMS[2] je open-source systém, ktorý je založený na knižnici React. Na správu obsahu využíva privátne GIT repozitáre, vďaka čomu výrazne znižuje požiadavky na výkon servera. Vďaka integrácii so službou GIT poskytuje široké možnosti verzovania a správy obsahu. V základnej verzii poskytuje však iba niekoľko základných nástrojov, ktoré môžu používatelia potrebovať. V prípade nedostatočnej funkcionality je možné vytvoriť si svoje vlastné rozšírenia (tzv. widget). Jeho hlavnou výhodou je rýchlosť, ktorú dosahuje vďaka knižnici React. Na obsluhu klientov poskytuje integráciu s generátormi statických stránok, ktoré taktiež využívajú React, ako je *Gatsby* alebo *Jekyll* alebo SSR[3] službami ako *NextJS*.

Squarespace

Squarespace[4] okrem správy obsahu umožňuje vystavanie celej webovej stránky pomocou jednoduchého procesu bez akejkoľvek potreby programovania. Poskytuje množstvo tém, ktoré sú optimalizované pre všetky druhy zariadení. Negatívom je uzavretosť systému, čo znamená, že sa používatelia musia uspokojiť so vstavanou funkcionalitou bez možnosti rozšírenia. Squarespace však ponúka široký sortiment služieb, vďaka čomu je možné vystavať akúkoľvek webovú stránku, od blogu až po jednoduchý e-shop. Taktiež poskytuje možnosť zakúpenia domény, vďaka čomu je možné spravovať celú stránku cez jednu službu.

4.3 Existujúce systémy pre správu tímu

Podobne ako na trhu s CMS, aj na trhu so systémami pre správu tímu je veľa dostupných možností. Neexistuje však žiadny systém, ktorý by tomuto trhu dominoval. Každý z týchto systémov má svoj unikátny prístup, čo pomáha používateľom vybrať si systém podľa ich potrieb.

[2] www.netlifycms.org

[3] Server Side Rendering - zobrazovaná stránka je vygenerovaná na strane servera a následne odoslaná klientovi

[4] www.squarespace.com

Basecamp

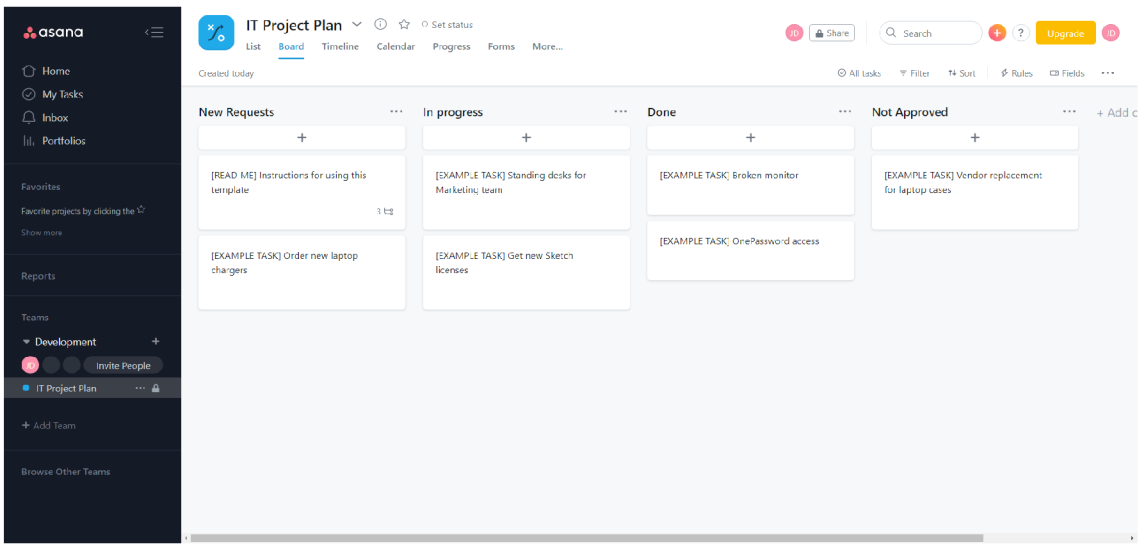
Basecamp[5] je jedným z pôvodných systémov pre správu tímu. Od svojho vzniku sa však veľmi zmenil a dnes ponúka obrovské množstvo funkcií, vďaka ktorým by sa ho dalo skôr označiť ako platformu, než systém na správu tímu. Napriek univerzálnosti tohto systému však neposkytuje také široké možnosti integrácie s inými službami.

Poskytuje diskusné fóra, online stretnutia a konferencie, zapisovanie času, sledovanie príchodu a odchodu z práce, stránky pre dokumentáciu tímu, individuálny a projektový kalendár a chat. Všetky tieto prostriedky však nie sú dostupné v rámci tímu, ale v rámci každého projektu. Basecamp je možné označiť za univerzálny systém, pretože eliminuje potrebu ostatných systémov v rámci pracovného postupu (workflow) tímu. Unikátnou vlastnosťou tohto systému je tiež služba *Klienti*, vďaka ktorej je možné kolaborovať s rôznymi klientami, pričom členovia tímu môžu nastaviť, čo daný klient vidí.

Asana

Asana[6] patrí medzi najpopulárnejšie systémy, ktoré sú dnes na trhu dostupné. Namiesto zamerania na správu tímu samotného sa profiluje ako nástroj na správu workflow v rámci tímu. Tento systém je vhodný pre tímy, ktoré pracujú kontinuálne a ich projekt nemá skutočný dátum ukončenia. Ponúka integráciu s viac ako 100 rôznymi službami, vrátane Adobe Creative Cloud, Slack, Microsoft Office, Jira a iné.

Základná štruktúra práce v systéme Asana pozostáva z tímu, ktorý má jeden a viac projektov, pričom každý projekt má úlohy. Každá úloha môže mať svoje podúlohy, prílohy a rôzne metadáta. Štruktúru je však možné upraviť podľa potrieb tímu, túto právomoc majú správcovia tímu. Pri vytvorení tímu je možné vybrať si z preddefinovaných šablón, no správcovia tímu môžu navrhnúť svoje vlastné šablóny. Tieto šablóny definujú štruktúru projektov, jednotlivých úloh a ich vzhľad. Šablóny sú rozdelené na základe odvetvia, v ktorom daný tím pracuje. Používateľské rozhranie je možné individuálne upraviť podľa vlastných potrieb. Prostredie systému Asana je na obrázku 4.2.



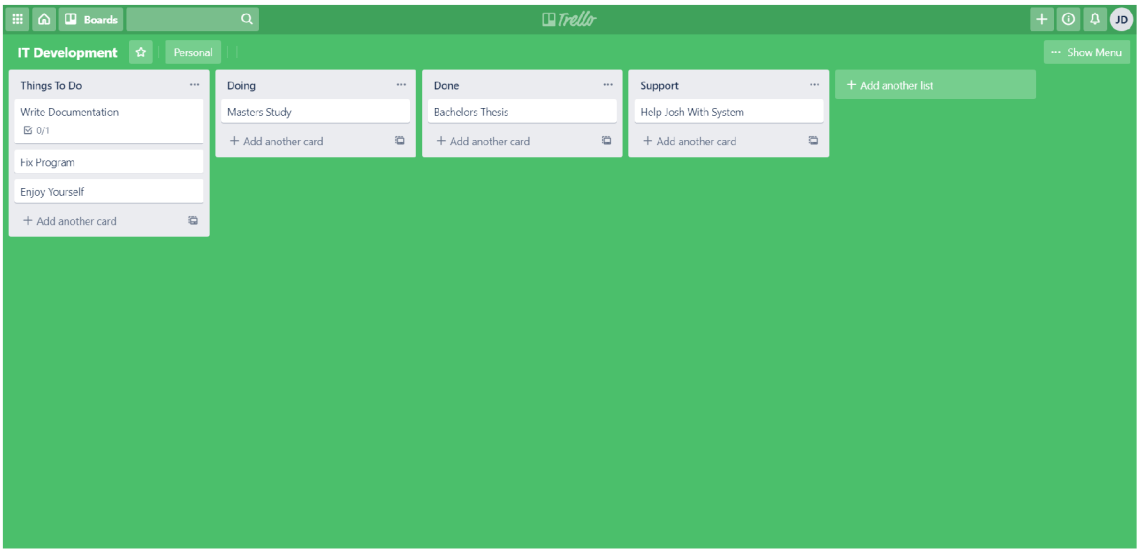
Obr. 4.2: Používateľské prostredie systému Asana

[5] www.basecamp.com/
[6] www.asana.com/

Trello

Trello[7] je v porovnaní s predchádzajúcimi systémami oveľa jednoduchší a jeho cieľová skupina sú skôr jednotlivci a menšie tímy. Každý používateľ však môže byť členom viacerých tímov, čo poskytuje výhodu pre ľudí, ktorí často menia klientov, alebo pre študentov. Trello taktiež poskytuje široké možnosti integrácie vrátane Slack, GitHub, GSuite a iné.

Štruktúra tohto systému je veľmi jednoduchá. Používateľské prostredie sa skladá zo zoznamov, ktoré sú v rámci tímu. Tieto zoznamy obsahujú karty. Karty predstavujú jednotlivé úlohy. Umožňujú pridávať detaily, komentáre, zoznamy podúloh alebo prílohy. Prostredie systému Trello je na obrázku 4.3.



Obr. 4.3: Používateľské prostredie systému Trello

4.4 Architektúra webových informačných systémov

Informačné systémy je možné rozdeliť do dvoch skupín na základe ich architektúry - monolitická a architektúra založená na mikroslužbách. Množstvo monolitických systémov, ako napríklad WordPress alebo Drupal, poskytuje taktiež alternatívu vo forme mikroslužieb. Porovnanie týchto architektúr je na obrázku 4.4.

Monolitická architektúra

Monolit je samostatný, zväčša mohutný kameň. Preto je vhodnou analógiou k systémom, ktoré obsahujú celú funkcionalitu v jednom. Monolitické systémy sa skladajú z jednej aplikácie, ktorá obsahuje backend, frontend a databázu. Pomocou tejto architektúry boli vyvinuté prvé informačné systémy. Ich hlavnými pozitívami je jednoduchosť vývoja, testovania a nasadenia. Tieto pozitíva sa však postupom času a postupným rastom aplikácie transformujú na negatíva. S aktualizáciami a pridávaním funkcionality jednotlivých komponentov aplikácie sa stále rozširuje repozitár obsahujúci kód pre celú aplikáciu. Toto môže pôsobiť

[7] www.trello.com/

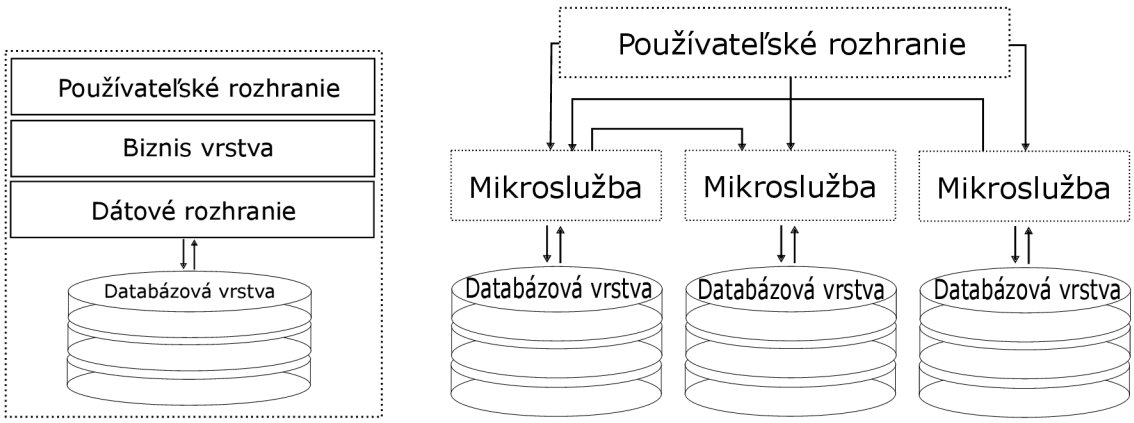
odstrašujúco pre nových členov tímu, ale taktiež spomalovať vývoj vďaka nárokom na IDE[8]. Ďalším negatívom je, že aj pri tej najmenšej aktualizácii jedného komponentu je potrebné znova zostaviť a nasadiť celú aplikáciu.

Architektúra založená na mikroslužbách

Architektúra založená na mikroslužbách sa snaží rozdeliť jednotlivé komponenty systému do samostatných služieb, ktoré medzi sebou komunikujú, bežne pomocou rozhraní postavených na protokole HTTP. Nejedná sa o nový prístup k vývoju aplikácií, ba práve naopak, princíp mikroslužieb vychádza z filozofie UNIX z roku 1978.

> Píšte také programy, ktoré robia jednu vec a robia ju dobre. [18]

Mikroslužby by na sebe mali byť čo najviac nezávislé, využívajú k tomu princíp delenia zodpovednosti[9]. Každá mikroslužba môže mať svoju vlastnú technológiu, proces vývoja, testovania a nasadenia. Výpadok jednej služby väčšinou neznamená znefunkčnenie systému. Jednou z najdôležitejších častí vývoja týchto systémov je analýza a rozdelenie na mikroslužby. Pri rozdelení na príliš mnoho mikroslužieb môže byť réžia komunikácie medzi nimi neúnosná.



(a) Monolitická architektúra (b) Architektúra založená na mikroslužbách

Obr. 4.4: Porovnanie architektúr informačných systémov

[8] Integrated Development Environment - vývojové prostredie, zväčša obsahujúce editor, prekladač, interpret a debugger

[9] každá mikroslužba má svoj účel

Kapitola 5

Návrh a architektúra aplikácie

Táto kapitola popisuje základné funkčné a nefunkčné požiadavky na aplikáciu, návrh jej architektúry, návrh používateľského rozhrania a v neposlednom rade výber technológií, ktoré budú využité v implementácii systému.

5.1 Aplikácia Contentu CMS

Aplikácia Contentu CMS poskytuje svojim používateľom základnú funkcionalitu CMS, teda pridávanie, správu a úpravu svojich stránok a článkov. Používatelia systému sa delia do troch kategórií – autor, korektor a manažér. Okrem iného je systém rozšírený o modul správy tímu a správu úloh v danom tíme. V rámci tímu je možné využiť schvaľovací proces, v ktorom používateľ zaradí článok do poradia a následne ho korektor alebo manažér môže publikovať. Správa úloh poskytuje možnosť tvorby a prideľovania úloh jednotlivým členov tímu.

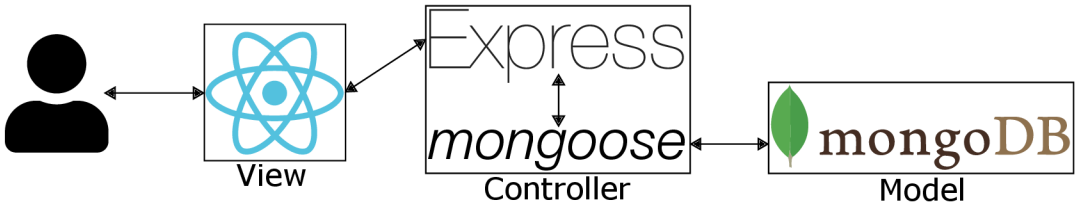
5.2 Technológie systému Contentu

Pre vývoj tejto aplikácie bol vybraný technologický balíček (technology stack) MERN, ktorý sa skladá z nasledujúcich technológií:

- **M**ongoDB, vzdialená (cloud) NoSQL databáza
- **E**xpress, backend framework pre tvorbu API
- **R**eact, knižnica pre tvorbu používateľských rozhraní
- **N**ode.js, prostredie pre beh JavaSript na serveri

Tento balíček je vhodný pre vývoj moderných webových aplikácií a má obrovskú komunitu vývojárov. Obrovskou výhodou je formát dát, s ktorým sa na samostatných vrstvách aplikácie pracuje. V každej vrstve aplikácie sa pracuje s formátom JSON (s výnimkou databázy, v ktorej sa využíva BSON, teda JSON dáta zakódované pomocou Base64.

MERN dodržiava MVC architektúru, pričom React reprezentuje View, Express reprezentuje Controller a mongoDB reprezentuje Model. Architektúra aplikácie je na obrázku 5.1.



Obr. 5.1: Architektúra systému

5.2.1 Frontend

Jedným z najdôležitejších aspektov pri výbere technológií bol vývoj frontendu. Nakoľko je Contentu komplexný systém cieľený na zjednodušenie práce pre širokú verejnosť, nemôže odrádzať používateľov svojou komplikovanosťou. Často sa budú meniť dáta, s ktorými používateľ priamo prichádza do kontaktu, a neustále aktualizácie obsahu pomocou zásahov do DOM alebo obnovovaním stránky by v dlhodobom hľadisku viedli ku zbytočnej komplexnosti kódu alebo znepríjemneniu práce so systémom pre cieľového klienta. Toto bolo hlavným dôvodom pre voľbu technológie React. Taktiež je nutné pripomenúť, že vďaka knižnici React bude možné využiť funkcie ES6[1] aj v starších prehliadačoch, bez nutnosti dodatočného prekladu pomocou nástrojov ako napríklad *Babel*.

5.2.2 Backend

Výber technológií pre backend bol jednoduchší v rámci počtu dostupných technológií, nakoľko som sa zameral na technológie, s ktorými už som mal skúsenosti, no o to bolo dôležitejšie klásť dôraz na jednotlivé rozdiely a ich dopad na aplikáciu. Na výber boli dve technológie s odlišným prístupom, PHP a Node.js. Po dôkladnej analýze som sa rozhodol zvoliť Node.js, pretože táto technológia pracuje asynchrónne a je riadená udalosťami. Takýto systém umožňuje prijímať niekoľkonásobne viac súbežných pripojení, nakoľko proces pri čakaní na zdroje neblokuje celé vlákno, ale vykonáva sa ďalej a po získaní potrebných zdrojov sa vykoná tzv. callback[2].

5.2.3 Databáza

Prvou otázkou pri výbere databázového systému bolo využitie relačnej alebo NoSQL databázy. Napriek tomu, že dáta v tomto systéme budú prevažne relačné, som sa rozhodol pre voľbu NoSQL databázy, konkrétne *MongoDB*. Z prvého pohľadu sa toto rozhodnutie môže zdať kontraproduktívne, dôvodom však bola vysoká škálovateľnosť a taktiež možnosť pracovať s dátami priamo, bez potreby využitia medzivrstvy vo forme objektovo relačného mapovania.

MongoDB

MongoDB je jeden z open-source zástupcov NoSQL databáz, ktoré na rozdiel od tradičných relačných databáz, využívajú dokumentovo orientovaný databázový model. MongoDB interne ukladá dáta vo formáte BSON (Binary JSON), formáte podobnému JSON (JavaScript Object Notation), ktorý umožňuje uchovávať viacero typov dát. BSON bol navrhnutý

[1] ECMAScript 6 - najnovšia špecifikácia jazyka JavaScript, ktorú vydáva zoskupenie ECMA

[2] Jedná sa o klasickú funkciu, ktorá sa spätne zavolá po dokončení inej funkcie. Využívajú sa najmä pri funkciách, ktoré získavajú dáta z databázy alebo iných asynchrónnych funkciách.

pre efektivitu a rýchlosť vyhľadávania jednotlivých dokumentov. Jedným z hlavných bodov záujmu je služba *Atlas*, ktorú poskytuje firma MongoDB (nepliesť s produktom MongoDB). Je to služba, ktorá umožňuje tvoriť vzdialené databázové servery, pri potrebe škálovania nevzniká žiadny čas nedostupnosti (downtime). Toto škálovanie je horizontálne, čo znamená, že sa pre zlepšenie výkonu alebo dostupnosti pridáva výpočetná kapacita systému vo forme ďalších serverov. Atlas umožňuje pomocou pár kliknutí vytvoriť globálnu sieť databázových serverov na platformách AWS (Amazon Web Services), Google Cloud Platform alebo Microsoft Azure, ktoré zaistia funkcionalitu a dostupnosť služby, kdekoľvek sa klienti služby nachádzajú.

5.3 Požiadavky na aplikáciu

Návrh systému bol vykonaný na základe analýzy dostupných riešení a konzultácii s profesionálnym žurnalistom. Je dôležité špecifikovať, že tento systém sa nesnaží byť najlepším systémom pre správu obsahu, ani najlepším systémom pre správu tímu. Cieľom tohto systému je poskytnutie príjemného prostredia na vykonávanie práce bez nutnosti využitia niekoľkých systémov a presunu dát medzi nimi.

Nefunkčné požiadavky aplikácie

Nefunkčné požiadavky nedefinujú správanie systému, ale definujú biznis ciele, ktoré v ideálnom prostredí tvoria konkurenčnú výhodu. Prehľad nefunkčných požiadavkov je v tabuľke 5.1.

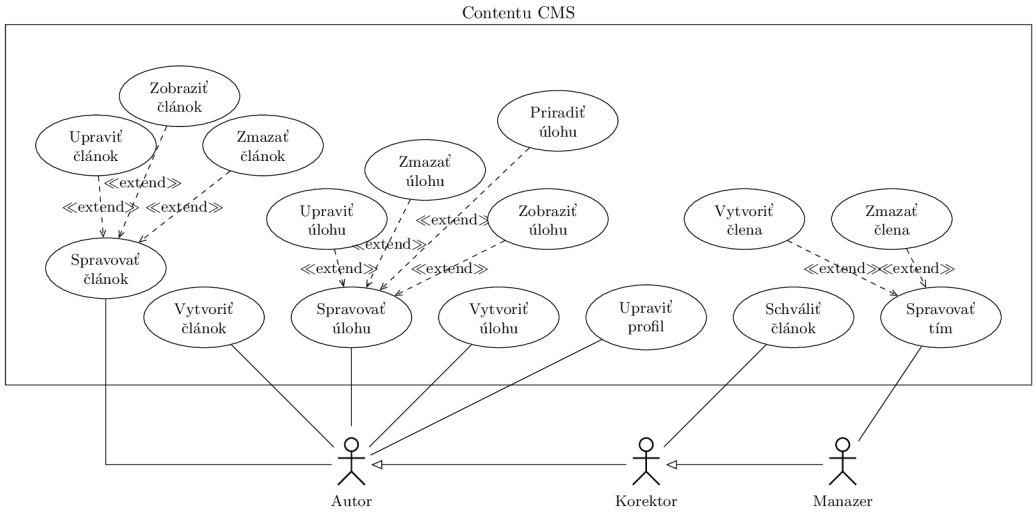
Požiadavka	Priorita
Systém musí byť modulárny a moduly musia mať rozdelenú zodpovednosť, aby systém vedel fungovať aj v prípade výpadku niektorého modulu	Stredná
Systém musí využívať databázu, ktorú je v prípade potreby možné vylepšiť alebo škálovať	Vysoká
Systém musí fungovať v moderných prehliadačoch (Chrome, Edge Chromium, Firefox)	Vysoká
Systém musí podporovať viacero používateľských rolí	Vysoká
Systém by mal byť intuitívny, prípadne obsahovať vysvetľujúce prvky	Stredná
Systém by mal používateľov upozorniť na nedostatočné heslo	Nízka
Systém by mal byť dostupný a prispôsobený pre všetky druhy zariadení	Vysoká

Tabuľka 5.1: Zoznam nefunkčných požiadaviek a ich priorita

Funkčné požiadavky aplikácie

Základné funkčné požiadavky na systém sú demonštrované pomocou diagramu prípadov použitia 5.2. V aplikácii delíme používateľov na troch aktérov, a to síce autor, korektor a manažér. Používateľský účet s rolou manažér vzniká registráciou. Následne môže manažér tvoriť účty pre ďalších členov tímu. Celá správa tímu je teda prenechaná osobe s právomocami manažéra. Samozrejme môže pracovať aj samostatne, bez potreby ďalších členov tímu. Manažér dedí všetku funkcionalitu od rolí korektor a autor. Používateľ s rolou korektor dedí funkcionalitu autora, no navyše je mu poskytnutá možnosť schvaľovať články, ktoré sú v prípade potreby v procese schvaľovania. Rola autora zahŕňa iba základnú funkcionalitu

aplikácie. Autor má možnosť tvoriť a spravovať články, a taktiež úlohy. Prehľad funkčných požiadaviek je zobrazený na obrázku 5.2.



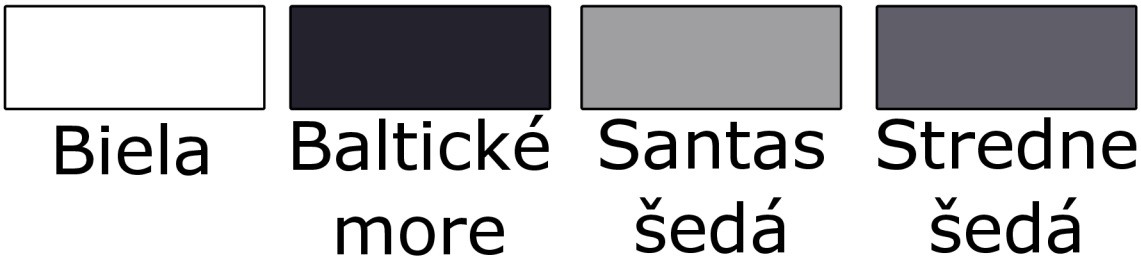
Obr. 5.2: Diagram prípadov použitia

5.4 Dizajn aplikácie

Dôležitým aspektom z pohľadu spokojnosti používateľov je dizajn. Rozloženie prvkov a ich farba môže mať obrovský dopad na používateľskú skúsenosť. Cieľom systému je vykonať prácu v čo najkratšom čase, bez pocitu obmedzenosti, alebo naopak otravnosti.

5.4.1 Farebná paleta

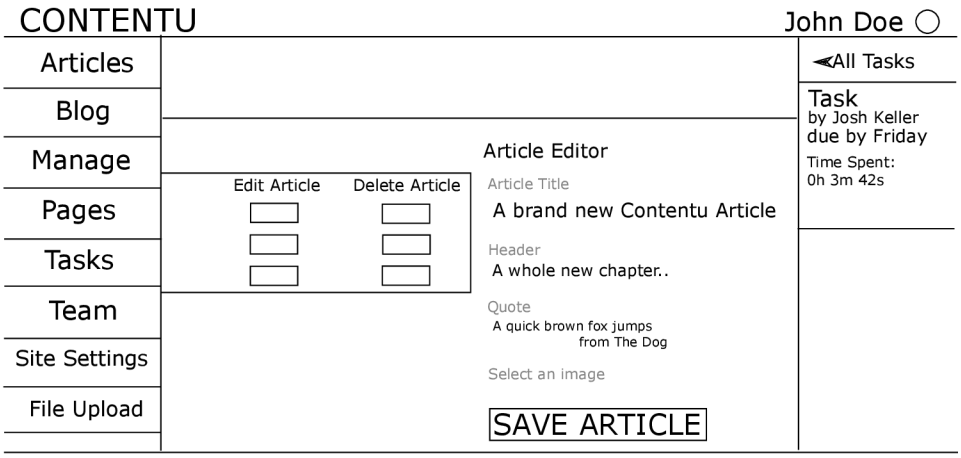
Farebná paleta bola navrhnutá za cieľom zvýraznenia kontrastu. Používateľ vie jasne od seba oddeliť jednotlivé komponenty iba pomocou vizuálnych prvkov. Systém využíva tmavé farby pre nemenné prvky systému a svetlé farby pre pracovný priestor. Tmavé farby, najmä tmavošedé odtiene, implikujú sofistikovanosť a minimálnosť. Systémové farby sú na obrázku 5.3.



Obr. 5.3: Farebná paleta aplikácie

5.4.2 Návrh používateľského rozhrania

Pri návrhu používateľského rozhrania je dôležité definovať cieľovú skupinu. Cieľovými používateľmi sú ľudia, ktorí už v minulosti využili systém pre správu obsahu. Návrh používateľského rozhrania sa inšpiruje aktuálnymi CMS, ktoré sú používateľsky prívetivé a dlhodobo overené. Aplikácia bude prevažne využívaná na počítačoch a notebookoch, no bude optimalizovaná aj pre mobilné zariadenia. Wireframe používateľského rozhrania je na obrázku 5.4.



Obr. 5.4: Návrh wireframe používateľského rozhrania systému Contentu

5.4.3 Navigácia v systéme

Tento systém obsahuje dva elementy na navigovanie po systéme. Jedna navigačná lišta je fixne v hornej časti stránky, obsahuje názov aplikácie a informácie o práve prihlásenom používateľovi, prípadne tlačítka pre prihlásenie. Druhú lištu, ktorá sa nachádza v ľavej časti stránky je možné podľa preferencií zúžiť, obsahuje samotnú navigáciu aplikácie, ktorá umožňuje prepínanie medzi jednotlivými časťami aplikácie.

5.4.4 Bočný panel s úlohami

Napravo od editora sa nachádza sekcia, v ktorej je možné prechádzať aktuálne úlohy a pracovať s nimi. Neumožňuje plnú funkcionalitu modulu úloh, slúži na prehliadanie a zmenu stavu počas práce na nejakom článku.

5.4.5 Hlavná časť aplikácie

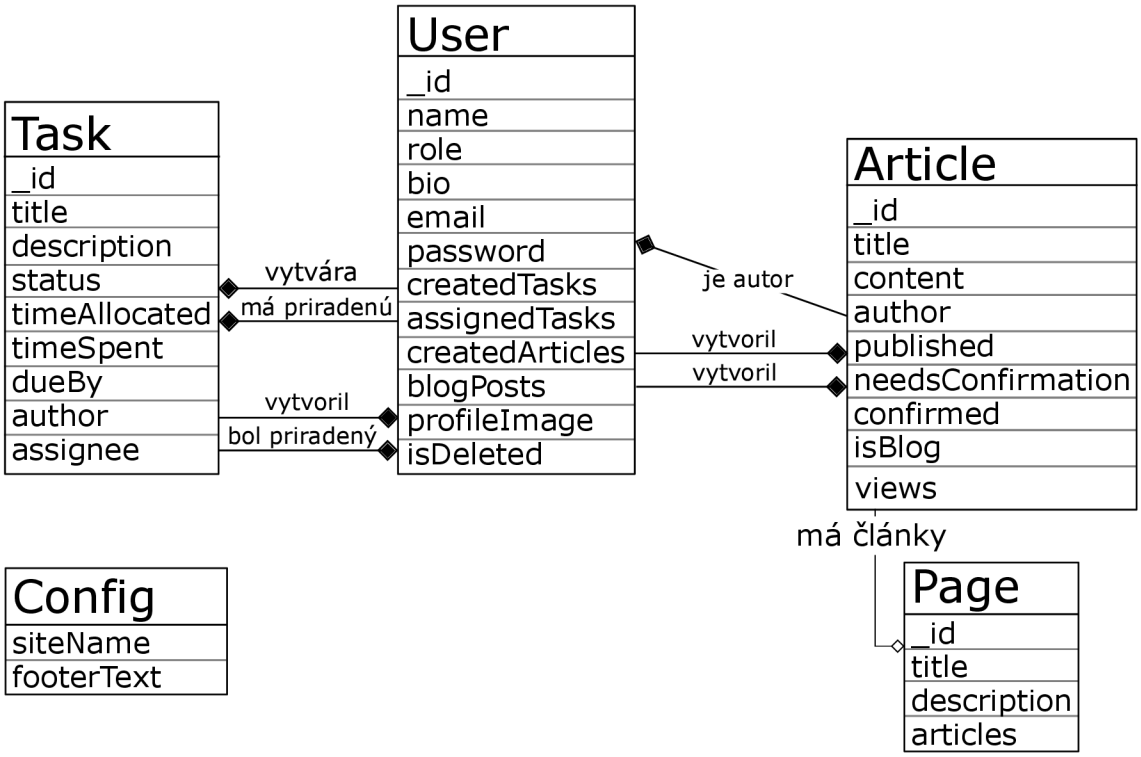
Hlavné okno aplikácie sa mení na základe sekcie, v ktorej sa používateľ v danej chvíli nachádza. Stránka *Články* obsahuje zoznam používateľských článkov a blokový WYSIWYG[3] editor. Stránka *Stránky* obsahuje zoznam stránok systému a formulár na ich tvorbu. Stránka *Správa* obsahuje všetky články v systéme a ich možnosť publikácie. Stránka *Tímový plán* obsahuje zoznam používateľských úloh a formulár na ich tvorbu. Stránka *Tím* obsahuje zoznam používateľov a formulár na ich tvorbu. Stránka *Nastavenia stránky* obsahuje predvoľby, ktoré predstavujú konfiguráciu stránky. Stránka *Správa Súborov* poskytuje rozhranie,

[3] z anglického „What You See Is What You Get“, teda „dostaneš to, čo vidíš“

cez ktoré je možné pracovať s obrázkami na serveri. Stránka *Náhľad stránky* poskytuje náhľad na stránku, ktorá sa zobrazí návštevníkom.

5.5 Návrh dátového modelu

Dáta v MongoDB Atlas sa ukladajú do tzv. zhlukov. Tieto zhluky sa delia na kolekcie, v ktorých už sú uložené samotné databázové objekty, teda dokumenty. Dokumenty majú svoje poddokumenty, ktoré by sa dali prirovnať ku stĺpcom tabuliek v relačnom modele. Poddokumenty predstavujú JSON objekty, ktoré môžu obsahovať ďalšie objekty alebo väzby na ne. V diagrame 5.5 sú tieto vzťahy vyjadrené pomocou vzťahov kompozície a agregácie. Vzťahy obsahujúce čierny kosoštvorec reprezentujú kompozitnú väzbu, teda väzbu, v ktorej existencia dcérskeho dokumentu závisí na existencii toho rodičovského. Agregácia je reprezentovaná prázdnym kosoštvorcom, teda dcérsky prvok môže existovať aj bez toho rodičovského.



Obr. 5.5: Diagram vzťahov entít systému

5.6 Používateľ

Zo štruktúry dát definovanej na obrázku 5.5 je zjavné, že používateľ je hlavnou časťou systému.

```
User {
    _id: ObjectId,
    name: String,
    role: String,
    bio: String,
    email: String,
    password: String,
    createdArticles: [ Article ],
    blogPosts: [ Article ],
    createdTasks: [ Task ],
    assignedTasks: [ Task ],
    profileImage: String,
    isDeleted: Boolean
}
```

Obr. 5.6: Formát databázového dokumentu typu User

5.6.1 Tvorba používateľských účtov

K tvorbe používateľských účtov sa využíva modul *Tím*. Pri prvotnom prístupe k systému je nutné zadať email, ktorý slúži ako manažérsky účet. Pomocou tohto účtu je následne možné v module *Tím* tvoriť ďalších používateľov, pričom je možné tvoriť aj ďalších používateľov v roli manažér. Na tvorbu používateľského účtu je potrebné zadať email, meno a rolu. V prípade, že sa jedná o používateľský účet s rolou *Autor*, je možné ho označiť ako dôveryhodného, vďaka čomu jeho články nemusia byť potvrdené manažérom alebo korektorom. Heslo k tomuto účtu je pseudonáhodne vygenerované a zaslané v potvrdzovacom emaile. Toto heslo si môže používateľ zmeniť po prvotnom prihlásení vo svojom používateľskom profile.

5.6.2 Prístup k používateľským dokumentom

Používatelia majú niekoľko obojsmerných väzieb na iné typy dokumentov. Táto väzba je realizovaná pomocou ukladania identifikátorov týchto dokumentov k príslušnému políčku používateľa. Jedná sa najmä o dokumenty typu Úloha, ktoré sú uložené v políčkach *createdTasks* a *assignedTasks*, a Článok, ktoré sú uložené v políčkach *createdArticles* a *blogPosts*.

5.6.3 Zmazanie používateľa

Zmazanie používateľa z databázy nie je možné, nakoľko sú požívatelia naviazaní na ostatné typy dokumentov. Namiesto toho sa pri tomto úkone ponechá dokument v systéme, no zmažú sa jeho údaje a nastaví sa značka *isDeleted*. Toto riešenie zachová v každom prípade konzistenciu dát a zamedzí zbytočným chybám systému.

5.7 Článok

Dokumenty typu Článok sú dôležitou súčasťou systému, nakoľko sú primárnym produktom pre koncových návštevníkov. Tieto dokumenty predstavujú okrem článkov v tomto systéme

aj blogové príspevky. Na tento účel slúži položka *isBlog*, ktorej pozitívna hodnota značí, že sa jedná o blogový príspevok.

```
Article {
    _id: ObjectId,
    title: String,
    content: String,
    author: User,
    views: Number,
    published: Boolean,
    needsConfirmation: Boolean,
    confirmed: Boolean,
    isBlog: Boolean
}
```

Obr. 5.7: Formát databázového dokumentu typu Article

5.7.1 Údaje o návštevníkoch

Systém umožňuje tvorcom sledovať návštevnosť ich článkov. K tomu slúží políčko *views* u dokumentov typu Článok.

5.8 Úloha

Dokumenty typu úloha sú hlavnou časťou modulu Tímový plán, nakoľko slúžia na plánovanie práce v tíme.

```
Task {
    _id: ObjectId,
    title: String,
    description: String,
    status: String,
    timeAllocated: Number,
    timeSpent: timeSpent,
    dueBy: String,
    author: User,
    assignee: User
}
```

Obr. 5.8: Formát databázového dokumentu typu Task

5.9 Stránka

Dokumenty typu stránka predstavujú objekty, ktoré sú zobrazené v navigačnom paneli výslednej stránky. Každá stránka má svoj názov, popis a zoznam článkov, ktoré sú na danej

stránke zobrazené. Zobrazenie článkov na týchto stránkach je regulované správcovským procesom, pričom sa zobrazia iba články, ktoré boli schválené a publikované.

```
Page {
    _id: ObjectId,
    title: String,
    description: String,
    articles: [ Article ]
}
```

Obr. 5.9: Formát databázového dokumentu typu Page

5.9.1 Zobrazenie článku na stránke

Pre zobrazenie a skrytie článku na stránke slúži modul Stránka. V tomto module si používateľ vyberie stránku, ktorú chce upraviť, a pomocou tlačidla *Manage* prejde na jej správu. Následne je mu poskytnutý zoznam dostupných článkov, pri ktorých si môže zvoliť, či je článok na stránke zobrazený alebo nie.

5.9.2 Blogové stránky

Blogové stránky sú špeciálnou inštanciou dokumentu stránka, pričom každý používateľ má svoju vlastnú blogovú stránku. Články na tejto stránke nepodliehajú správcovskému procesu, teda nemusia byť schválené správcom, aby k nim mohli pristupovať návštevníci.

5.10 Konfigurácia

Konfiguračný dokument slúži k vyplneniu údajov o webovej stránke vytvorenej v systéme Contentu. Umožňuje upravovať názov stránky zobrazený v hlavičke, a taktiež text, ktorý je zobrazený v päte stránky.

```
Config {
    siteName: String,
    footerText: String
}
```

Obr. 5.10: Formát databázového dokumentu typu Config

Kapitola 6

Implementácia

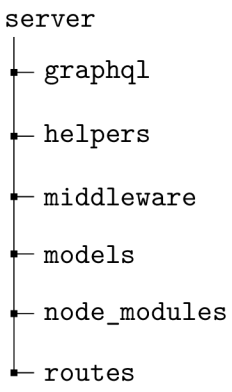
Táto kapitola popisuje implementáciu jednotlivých častí systému *Contentu*. Čitateľovi bude vysvetlená štruktúra projektu, spôsob inštalácie a integrácia jednotlivých modulov. Implementácia aplikácie vychádza z vykonanej analýzy požiadaviek, ich následnej špecifikácie a vytvorenia návrhu systému v kapitole 5. Prebiehala v iteráciách, pričom prvým cieľom bolo implementovanie všetkých častí systému aj za cenu ich zníženej funkcionality. Táto technika sa nazýva MVP[1]. V nasledujúcich iteráciách sa postupne pridávala navrhnutá funkcionalita.

6.1 Štruktúra projektových súborov

Celý systém je rozdelený do dvoch služieb. Prvou službou je server, ktorý sa stará o správu obsahu a autentifikáciu používateľov. Nachádza sa v adresári *server*. Druhou službou je webová aplikácia vytvorená prostredníctvom prostredia *create-react-app*, ktorá slúži ako používateľské rozhranie ku práci so systémom. Tá sa nachádza v adresári *client*. Adresárová štruktúra serverovej služby je na obrázku 6.1 a klientskej služby na obrázku 6.2.

6.1.1 Štruktúra adresára server

Serverová vrstva aplikácie je usporiadaná nasledovne:
Adresár *graphql* - obsahuje GraphQL schéma a resolvery pre požiadavky a mutácie
Adresár *helpers* - obsahuje pomocné funkcie
Adresár *middleware* - obsahuje pomocné funkcie, ktoré sa vykonávajú medzi prijatím požiadavky serverom a jej spracovaním
Adresár *models* - obsahuje modely, ktoré definujú štruktúru dokumentov v databáze
Adresár *node_modules* - obsahuje NodeJS balíčky
Adresár *routes* - obsahuje funkcie na spracovanie po-



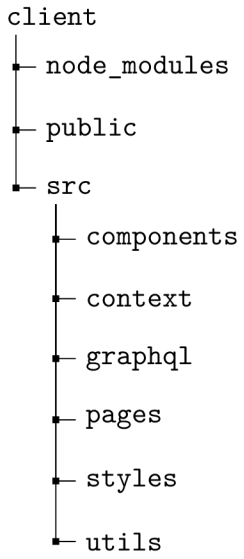
Obr. 6.1: Adresarová štruktúra služby server

[1]Minimum Viable Product - technika vývoja, pri ktorej je vyvinutá minimálna funkcionalita aplikácie, ktorá dokáže presvedčiť o prínose aplikácie

žiadaviek od koncových klientov

6.1.2 Štruktúra adresára client

Súbory pre klientskú vrstvu sú usporiadané nasledovne:
Adresár *node_modules* - obsahuje NodeJS balíčky
Adresár *public* - obsahuje súbory, ktoré sú odosielané klientovi
Adresár *src* - obsahuje zdrojové kódy klientskej aplikácie
Adresár *src/components* - obsahuje znovupoužiteľné komponenty aplikácie
Adresár *src/context* - obsahuje správu stavu aplikácie
Adresár *src/graphql* - obsahuje definície GraphQL požiadaviek a mutácií
Adresár *src/pages* - obsahuje jednotlivé stránky aplikácie
Adresár *src/styles* - obsahuje CSS súbory
Adresár *src/utils* - obsahuje pomocné funkcie (Apollo klient pre komunikáciu so serverom)



Obr. 6.2: Adresarová štruktúra služby client

6.2 Balíčky tretích strán serverovej aplikácie

Serverová aplikácie reprezentuje vrstvu controller v MVC architektúre. Jej cieľom je prijímanie požiadaviek, ich transformácia na príkazy pre model a odosielanie relevantných odpovedí. V serverovej aplikácii boli využité nasledovné balíčky.

Express

Express je najpopulárnejší webový aplikačný rámec v prostredí Node.js. Umožňuje jednoducho a rýchlo vytvárať serverové aplikácie a API. Express je minimalistický, no vďaka veľkej komunite vývojárov je možné ďalšiu funkcionalitu jednoducho doinštalovať pomocou modulov dostupných cez npm. Express slúži ako HTTP server, no poskytuje možnosť rozšírenie pomocou middleware. Tento middleware typicky vykonáva funkciu aplikačného servera. Umožňuje tvorbu obslužných rutín pre rôzne endpointy a ich príslušné HTTP metódy.

Apollo-server-express

Tento open-source balíček slúži ako GraphQL server v aplikačnom rámci Express. Taktiež v sebe obsahuje rozhranie *gql*, ktoré umožňuje definovať schému, požiadavky a mutácie.

Bcryptjs

Balíček bcryptjs umožňuje hašovanie dôležitých údajov pomocou jednosmerného hašovacieho algoritmu, ktorý je založený na symetrickej šifre zvanej Blowfish. Hašované údaje majú podobu odtlačku, ktorý má fixnú dĺžku a nie je možné ho rozšifrovať. Tento balíček umožňuje variabilnú dĺžku soli, vďaka čomu je výrazne zvýšená bezpečnosť hašovaných dát.

GraphQL

Balíček GraphQL je open-source referenčná implementácia rozhrania nad jazykom GraphQL v prostredí Node.js.

Nodemailer

Balíček nodemailer slúži na odosielanie emailových správ z prostredia Node.js. Poskytuje rozhranie nad protokolom SMTP[2], pričom umožňuje odosielať správy vo formáte HTML, čistého textu alebo príloh. Tento balíček využíva najnovšiu špecifikáciu JavaScriptu a neobsahuje žiadne závislosti, čo výrazne zvyšuje jeho bezpečnosť.

Jsonwebtoken

Balíček jsonwebtoken poskytuje JavaScript rozhranie nad JWT. Umožňuje tvorbu a podpisovanie tokenov, pričom je možné definovať hašovací algoritmus a kľúč, ktorým je tento token podpísaný.

Mongoose

Mongoose je balíček poskytujúci rozhranie nad natívnym MongoDB ovládačom v prostredí Node.js. Mongoose definuje schému databázy za účelom okamžitej validity dát. Táto schéma reprezentuje štruktúru dokumentu, jeho predvolené hodnoty a validátory dát. Taktiež poskytuje modely jednotlivých typov dokumentov, ktoré slúžia ako vysoko úrovňové rozhranie nad definovanou schémou. Tie umožňujú vykonávanie CRUD funkcionalitu nad dokumentmi v databáze.

6.3 Balíčky tretích strán klientskej aplikácie

Klientská aplikácia reprezentuje vrstvu view v MVC architektúre. Jej cieľom je zobrazovanie relevantných dát používateľom a poskytnutie rozhrania nad stavom aplikácie. V klientskej aplikácii boli využité nasledovné balíčky.

@apollo/react-hooks

Balíček @apollo/react-hooks zjednodušuje požiadavky a mutovanie dát na GraphQL API vo funkcionálnych React komponentoch. Bez využitie tohto balíčka by bolo nutné v každom komponente, ktorý požaduje nejaké dáta, importovať Apollo klient a využívať jeho metódy `client.query()` alebo `client.mutate()`.

[2]Simple Mail Transfer Protocol

Apollo-client

Apollo-client je balíček slúžiaci k správe stavu JavaScript aplikácií, ktoré získavajú dáta z GraphQL API. Poskytuje rozhranie nad metódami, pričom sa stará o získanie dát, ich uloženie do vyrovnávacej pamäte a aktualizáciu používateľského rozhrania.

Apollo-cache-inmemory, apollo-link-error, apollo-upload-client

Tieto balíčky slúžia ako rozšírenie Apollo klienta. Apollo-upload-client rozširuje funkcionalitu klienta, umožňuje nahrávať súbory na server skrz GraphQL API. Apollo-link-error poskytuje rozšírené možnosti úpravy chybových výpisov, ktoré sa vracajú zo strany servera. Je to middleware, ktorý tieto chyby zachytí, vykoná preddefinované úpravy na základe nastavení a odošle chybu klientovi. Apollo-cache-inmemory je odporúčaná implementácia pre správu cache v prostredí Apollo.

React-dropzone

Balíček react-dropzone slúži na výber súborov v prostredí React. Poskytuje širokú škálu možností, pričom podporuje výber pomocou základného prehliadača súborov poskytovaného operačným systémom alebo pomocou HTML5 funkcie *drag and drop*, ktorá umožňuje pretiahnuť súbory do HTML elementu. Tieto súbory je následne možné odoslať pomocou formulárov alebo iným spôsobom spracovať.

6.4 Požiadavky a mutácie definované serverovou časťou

Serverová časť poskytuje dáta pomocou GraphQL požiadaviek a umožňuje meniť stav systému pomocou mutácií. Dôkladný zoznam dostupných požiadaviek je dostupný v prílohe A a zoznam mutácií je dostupný v prílohe B. Server očakáva tieto požiadavky a mutácie na adrese `/graphql`.

6.5 Databázový systém

Hlavným cieľom tohto systému je tvorba, správa a zverejňovanie informácií. Z toho dôvodu je dôležité vybrať optimálny spôsob ukladania informácií. Prvotným plánom bolo ukladanie komplexnejších dát, ako sú napríklad články, do súborov na serveri a ukladanie jednoduchších dát, ako sú metadáta článkov, do Redis databázy. Takýto prístup sa však pre tento systém ukázal ako kontraproduktívny, nakoľko vzniklo zbytočné zaťaženie servera, čo sa nakoniec prejavilo znížením výkonu aplikácie.

Hlavným úložiskom tohto systému je databáza MongoDB. Pri implementácii som sa rozhodoval, či budem využívať natívny ovládač MongoDB alebo balíček tretích strán. NodeJS obsahuje niekoľko knižníc, ktoré umožňujú komunikáciu s touto databázou. Pre tento projekt som vybral balíček *mongoose*, ktorý využíva natívny MongoDB ovládač, no rozširuje ho o funkcionalitu modelovania dát, definovania schém a umožňuje spätné volania po získaní dát.

Prístup k databáze je následne relatívne jednoduchý. Do projektu stiahneme balíček *mongoose* pomocou príkazu `npm -i mongoose` a následne v programe importujeme pomocou príkazu `const mongoose = require("mongoose")`. V momente spustenia programu

sa vykoná konštruktor tohto balíčka, a vytvorí objekt, ktorý môže byť využitý ku komunikácii s databázou. Najprv je potrebné sa pripojiť, čo sa vykoná pomocou metódy `mongoose.connect(url, options)`.

6.6 Autentifikácia a autorizácia

Systém z princípu umožňuje prácu iba prihláseným používateľom. Autentifikácia je dôležitým zabezpečovacím prvkom systému, nakoľko overuje, že komunikácia medzi klientom a serverom nebola napadnutá alebo zneužitá treťou stranou. O proces autentifikácie sa stará autentifikačný resolver. Pokiaľ je používateľ prihlásený, jeho token sa nachádza v lokálnom úložisku. Tento token sa odosiela pri každej požiadavke, pričom sa nachádza v autentifikačnej HTTP hlavičke *Authorization*. V opačnom prípade najprv vyhľadá v databáze používateľa podľa emailu a pokiaľ existuje, uloží ho do premennej `user`, následne overí heslo používateľa pomocou metódy `bcrypt.compare(password, user.password)`. Pokiaľ niektorý z týchto krokov zlyhá, používateľovi je odoslaná príslušná chybová hláška. V opačnom prípade je vygenerovaný JWT prístupový kľúč s platnosťou 12 hodín, ktorý je odoslaný klientovi. Na strane klienta sa tento prístupový kľúč uloží na lokálne úložisko. V prípade opakovaného prístupu na strane klienta sa overí platnosť kľúča, pokiaľ je platný, používateľ ostane prihlásený. Na prenos je využitá schéma *Bearer*, ktorá slúži na autentifikáciu pomocou tokenov. Server obsahuje middleware, ktorý overí existenciu autentifikačnej hlavičky. V prípade jej existencie overí platnosť JWT a nastaví používateľský identifikátor do hlavičky *userID* a požiadavku deleguje na ďalšie spracovanie. Overenie právomocí je vykonané v každej požiadavke samostatne, nakoľko niektoré požiadavky vyžadujú vyššie poverenie, ako ostatné.

6.7 Smerovanie

O smerovanie v aplikácii sa stará knižnica *BrowserRouter* z balíčka *react-router-dom*, ktorý poskytuje komponent *Route*. Nad týmto komponentom bola postavená nadstavba *PrivateRoute*, ktorá zaisťuje, že k danej stránke má prístup len oprávnený používateľ.

Komponent PrivateRoute

Táto nadstavba nad komponentom Route slúži na autorizáciu prístupu používateľa k modulu. V prvom rade sa overuje prítomnosť a platnosť tokenu v lokálnom úložisku. Ak je platný, získajú sa zo serveru dáta potrebné pre chod aplikácie. Následne sa overí, či je používateľ autorizovaný ku danému modulu. Toto overenie prebieha na základe porovnania používateľskej role s preddefinovaným zoznamom ciest a ich povolených rolí. Ak užívateľ nie je prihlásený alebo má uložený neplatný token, je presmerovaný na prihlásenie. Ak je používateľ prihlásený, no nie je autorizovaný, je presmerovaný na koreňový adresár služby. Ak je prihlásený a autorizovaný, služba pokračuje na požadovaný modul.

6.8 Správa stavu aplikácie

Samotná knižnica React neposkytuje veľa možností správy stavu. Jediným spôsobom je odosielanie stavu rodičovského komponentu jeho následníkom. Toto sa deje pomocou tzv.

props[3]. To stačí pre jednoduché aplikácie, ktoré nemajú veľa stavov, a komponenty medzi sebou nepotrebujú komunikovať. V prípade komplexnejších aplikácií však prichádza aj nutnosť prístupu väčšiny komponentov ku globálnemu stavu, a preto boli vyvinuté rozšírenia ako *Redux* alebo *Context*. Táto aplikácia využíva na správu stavu *Context API*.

Aplikácia obsahuje 3 podstavy, *AuthState*, ktorý zodpovedá za správu používateľa, *BadgeState*, ktorý zodpovedá za zobrazovanie upozornení a *EditorState*, ktorý zodpovedá za správu WYSIWIG editora. Každý podstav je tvorený z troch častí:

- Context - komponent vytvorený pomocou metódy `createContext()` knižnice React, ktorý obaľuje komponenty, v ktorých má byť globálny stav dostupný
- State - definuje premenné systému, ich počiatočný stav a metódy, ktorými je možné vytvárať udalosti pre reducer
- Reducer - prijíma udalosti a na ich základe upravuje stav

6.9 Správa súborov

GraphQL špecifikácia nedefinuje spôsob, ako nahrávať súbory. Pri vývoji bol použitý balíček *Apollo*[4], ktorý má nadstavbu *Upload*. Server definuje dve mutácie, *uploadFile* a *deleteFile*. Po prijatí súboru sa súbor uloží do adresára *files*, z ktorého sú následne prístupné cez požiadavku *getFiles*, alebo priamo cez URL *server.tld/files/filename*. Na strane servera sa súbory ukladajú do adresára *files*.

6.10 Nasadenie aplikácie

Vďaka rozdeleniu systému na mikroslužby sú možnosti nasadenia systému takmer nekonečné. Systém je možné celý nasadiť na jeden server, kde mikroslužby bežia súčasne, alebo každú mikroslužbu na samostatný server. Tým je umožnená vysoká škálovatelnosť aplikácie, čo v prípade nedostatočného výkonu v budúcnosti výrazne zníži náklady spojené s úpravou aplikácie.

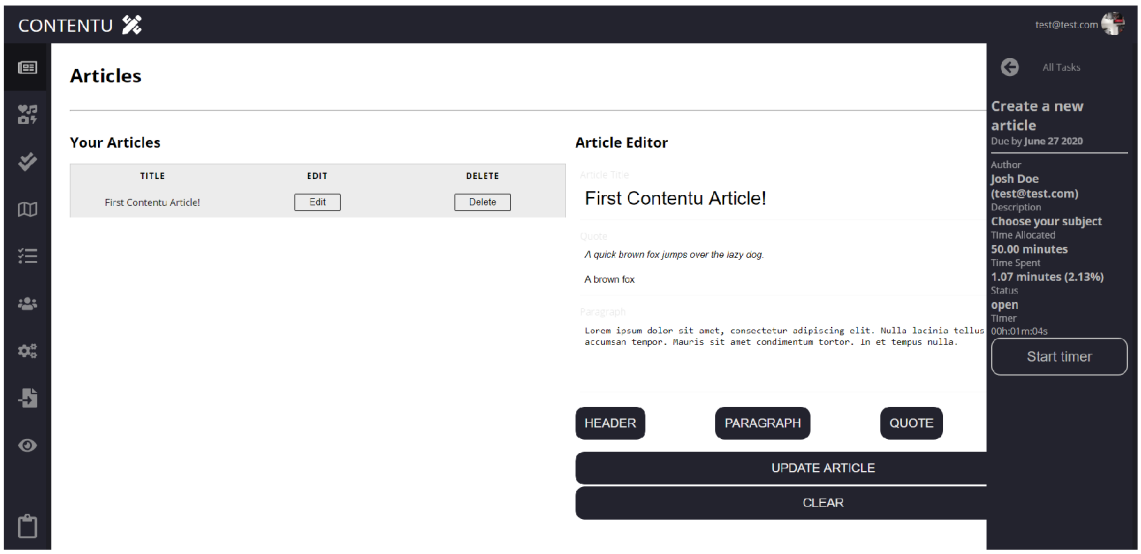
Pre nasadenie aplikácie som sa rozhodol využiť službu Heroku[5]. Ich základný balíček je zadarmo a postačí pre aplikácie s nižším počtom používateľov. Pre študentov taktiež ponúkajú zadarmo vyšší balíček, ktorý zlepšuje parametre servera a pridáva možnosť využitia SSL certifikátu, vďaka čomu je zaistená bezpečná komunikácia. Samotné nasadenie je potom relatívne jednoduché. V ovládacom paneli Heroku sa vytvorí nový projekt, cez príkazový riadok sa do neho nahrá repozitár s projektom, ktorý obsahuje skript *heroku-postbuild*. Skrz ovládací panel projektu sa aplikácia zostaví a zverejní. Po nasadení sa vyskytla chyba, ktorá spôsobovala nefunkčnosť aplikácie. Heroku obsahuje svoj vlastný smerovač, ktorý nie je kompatibilný so smerovačom knižnice React. Prístup k iným stránkam ako tej koreňovej skončil HTTP chybou *500 Interal Server Error*. Dočasným riešením je súbor *static.json* v koreňovom adresári servera, ktorý je nadradený nad pôvodným smerovačom, a presmerováva všetky požiadavky na koreňovú stránku aplikácie.

Ďalším problémom bola nefunkčnosť niektorých funkcií knižnice React, čo sa vyriešilo použitím správcom závislostí *create-react-app-buildpack*[6].

[3] skrátený názov odvodený z anglického properties, teda vlastnosti
[4] www.apollographql.com/docs/apollo-server/
[5] www.heroku.com
[6] www.elements.heroku.com/buildpacks/mars/create-react-app-buildpack

6.11 Konečný stav systému

Výsledkom tejto práce je systém, ktorý je implementovaný pomocou moderných webových technológií a princípov. Tento systém je otestovaný, a pripravený na produkčné testovanie. Pohľad na hlavný modul systému je na obrázku 6.3.



Obr. 6.3: Pohľad na modul Články v systéme Contentu

Kapitola 7

Testovanie

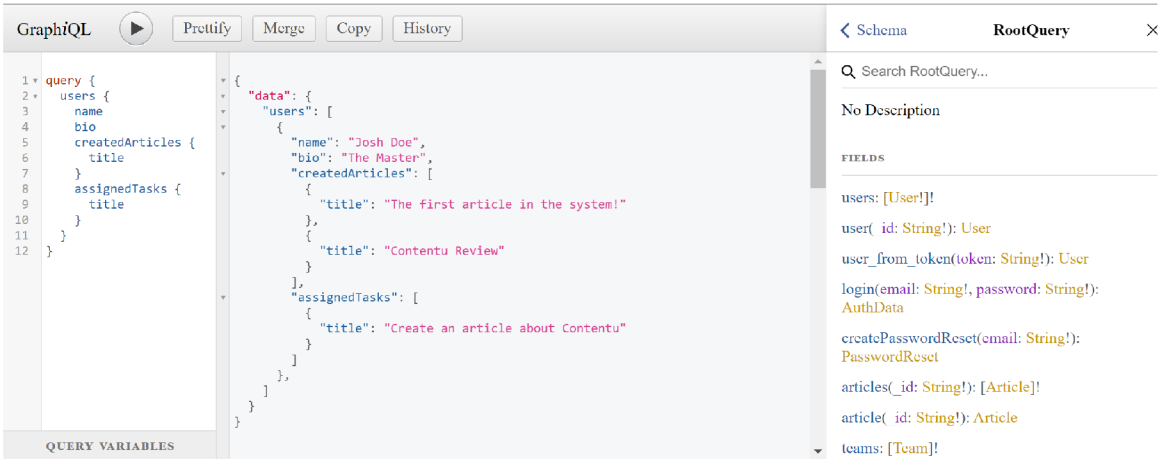
Táto kapitola popisuje testovanie, ktoré je rozdelené do dvoch častí. V prvej časti bude popísané testovanie vývojárom, v druhej časti bude popísané testovanie na skutočných používateľoch.

7.1 Testovanie vývojárom

Na strane vývojára bolo potreba testovať obe mikroslužby systému. Testovanie serverovej časti prebiehalo manuálne, pomocou testovania požiadaviek a mutácií. Testovanie klientskej časti prebiehalo automatickými testami.

Testovanie serverovej časti

Testovanie serverovej časti aplikácie bolo menej časovo náročné, nakoľko GraphQL poskytuje grafické rozhranie *GraphiQL*, ktoré umožnuje testovanie API a automaticky tvorí jeho dokumentáciu. Prostredie nástroja GraphiQL je na obrázku 7.1.



Obr. 7.1: Nástroj GraphiQL

V prípade mutácií však bolo potreba využiť nátroj GraphQL Playground[1], nakoľko aplikácia vyžaduje overenie pomocou užívateľského tokenu, a nástroj GraphiQL nepodporuje špecifikáciu HTTP hlavičiek.

[1] www.apollographql.com/docs/apollo-server/testing/graphql-playground/

Testovanie klientskej časti

Testovanie klientskej časti bolo vykonané za pomoci nástroja *Jest*. Jest je open-source testovací framework, ktorý funguje bez konfigurácie s väčšinou projektov napísaných v jazyku JavaScript. Po vytvorení každého nového komponentu bola pre neho vytvorená séria testov. Takýmto spôsobom je možné overiť funkčnosť nového komponentu, ale aj to, že žiadnym spôsobom nebola ovplyvnená funkčnosť zbytku aplikácie.

V prostredí *create-react-app* je Jest predinštalovaný. Stačí vytvoriť súbory s príponou **.test.js*, a spustením príkazu `npm test` sa spustí testovacia sada, ktorá najprv vyhľadá všetky **.test.js* súbory, a vykoná definované testy. Aplikáciu je potreba otestovať samostatne a nezávisle od ostatných častí systému. Je preto treba využiť Jest metódu *mock*, pomocou ktorej môžeme definovať statické dáta, ktoré budú využité na testovanie. Týmto spôsobom zabránime narušeniu testovania vonkajšími vplyvmi (nedostupnosť API, pomalé pripojenie...) a zaistíme, že testovanie je vždy validné.

7.2 Používateľské testovanie

Testovanie na používateľoch prebiehalo pomocou metódy čiernej skrinky. To znamená, že používateľ je oboznámený s funkcionalitou aplikácie, no nepozná vnútorný princíp fungovania. Pôvodným plánom bolo dvojfázové testovanie používateľov. Po prvej fáze implementácie, teda po vytvorení MVP, bolo naplánované testovanie metódou premýšľania nahlas. Cieľom tejto fázy bolo zistenie nedostatkov v používateľskom rozhraní a plynulosti systému. V druhej fáze bolo naplánované testovanie za pomoci vytvoreného testovacieho protokolu. Toto však skomplikovala aktuálna svetová situácia, ktorá bola z veľkej časti ovplyvnená celosvetovou pandémiou COVID-19.

Testovanie bolo potreba upraviť, aby mohlo byť vykonané cez internet. Na komunikáciu som vybral program Discord[2], ktorý je zadarmo a poskytuje aj možnosť zdieľania obrazovky. Každý subjekt bude samostatne vykonávať nasledujúci testovací protokol, následne dostanú minútu na jeho prečítanie a potom začne testovanie, pri ktorom sa bude zaznamenávať čas potrebný na každú úlohu. Každý subjekt dostane prihlasovacie údaje k ich vlastnému profilu. Tieto profily sú nastavené na rolu manažéra, čím je subjektom sprístupnená plná funkcionalita systému. Príliš dlhý čas medzi jednotlivými úlohami môže znamenať, že systém nie je prehľadný alebo dostatočne intuitívny. Čas bude zaznamenávaný od bodu číslo 1:

0. Pripravte si ľubovoľný obrázok na plochu vášho počítača
1. Prihláste sa do systému s poskytnutými údajmi
2. Vytvorte úlohu, kde názov je vaše meno, je na ňu vyhradených 50 minút, splnená musí byť do 15.5.2020 a je určená pre vás.
3. Prejdite na stránku Nahraj súbory (Upload Files) a nahrajte pripravený obrázok
4. Prejdite na stránku Články (Articles) a z panelu úloh vyberte vytvorenú úlohu a spustite časovač.
5. Vytvorte nový článok, kde názov je vaše meno, a obsahuje 1 nadpis, 1 textové pole a 1 citát s ľubovoľným obsahom.

[2] www.discordapp.com

6. Pridajte obrázkový blok a zvoľte váš obrázok, článok uložte.
7. Upravte tento článok, zmažte textový blok a článok uložte.
8. V paneli úloh zastavte časovač.
9. Prejdite na stránku Stránky (Pages).
10. Vytvorte novú stránku, kde názov je vaše meno a popis je ľubovoľný.
11. Priraďte tejto stránke váš vytvorený článok.
12. Prejdite na stránku Správa (Manage), nájdite svoj článok a zverejnite ho.
13. Zobrazte si náhľad článku.
14. Prejdite na svoj profil a zmeňte si heslo.
15. Odhláste sa.

Výsledky testovania

Testovania sa zúčastnili štyri subjekty, z ktorých traja sú študentmi vysokej školy a jeden je novinár. Výsledky testovania jednotlivých subjektov sú prezentované v tabuľke 7.1. Uvádzaný je celkový čas subjektov od počiatku testovania.

Číslo úlohy	Subjekt 1	Subjekt 2	Subjekt 3	Subjekt 4
1	00:04	00:08	00:05	00:07
2	00:30	00:40	00:31	00:34
3	00:47	01:00	00:44	00:46
4	00:56	01:12	00:54	00:51
5	01:14	01:30	01:12	01:12
6	01:25	01:39	01:24	01:27
7	01:37	01:50	01:34	01:40
8	01:42	01:56	01:40	01:47
9	01:47	02:02	01:50	02:00
10	02:00	02:30	01:59	02:07
11	02:12	02:45	02:10	02:15
12	02:31	03:00	02:29	02:33
13	02:36	03:05	02:40	02:40
14	03:00	03:42	3:10	03:12
15	03:10	04:00	3:15	03:17

Tabuľka 7.1: Výsledky testovania

Z výsledkov testovania vyplýva, že žiadny zo subjektov nemal väčšie problémy s daným testovacím protokolom a fungovaním systému. Treba však brať na zreteľ, že sa jedná o technicky zdatných používateľov. Odchýľky vznikli väčšinou pri vypisovaní políčok, pričom nebol špecifikovaný obsah a niektorým subjektom trvalo zadávanie údajov dlhšie.

Spätná väzba

Po vykonaní testovania boli subjekty požiadané o poskytnutie spätnej väzby a ich názoru na systém, po dizajnovej, ale hlavne funkčnej stránke. Tieto výsledky sú zhrnuté v tabuľke 7.2

Subjekt	Spätná väzba
1	Aplikácia je prehľadná, no je treba si ju preklikať, aby sa človek v nej zorientoval. Po úvodných pár minútach to bolo v poriadku. Nepáčil sa mi dvojsĺtpcový dizajn, ale dá sa na to zvyknúť. Taktiež sa mi nepáči, že sa články ukladajú aj pri preklikávaní a ostatné formuláre nie.
2	Systém sa mi páči, je rýchly, pekný a funkčný, nemám mu čo vytknúť.
3	Páčil sa mi jednoduchý a čistý dizajn a intuitívnosť v systéme pre používateľa. Systém funguje, nemám čo vytknúť.
4	Bočný panel s úlohami by som rád mal v celom systéme, nielen na stránke články. Okrem toho sa mi páčil dizajn, systém fungoval responzívne a nenarazil som na žiadne chyby. Páči sa mi, že je optimalizovaný pre mobilné zariadenia.

Tabuľka 7.2: Spätná väzba ku systému Contentu

Kapitola 8

Záver

Hlavným cieľom tejto práce bolo vypracovanie analýzy, návrhu a implementácie systému, ktorý zjednoduší organizáciu a prácu ako jednotlivcom, tak aj tímom.

V prvej časti práce bol stručne popísaný vznik webu, jeho história a jeho technológie. Boli vysvetlené dôležité pojmy a princípy fungovania World Wide Webu. Následne boli analyzované existujúce riešenia a ich architektúra.

Na základe analýzy existujúcich riešení a predstavených technológií boli popísané požiadavky na službu a jej návrh. Návrh sa skladal z popisu architektúry systému, jeho častí a návrhu používateľského rozhrania. Bol definovaný dátový model, ktorý obsahoval dôkladný popis jednotlivých druhov dokumentov systému a ich interakcie medzi sebou. Rozhodnutia o výbere technológií a návrhových vzorov boli vykonané už počas tvorby návrhu. Tento návrh teda presne definoval implementačné detaily. Dôsledkom toho som počas implementácie narazil na niekoľko úskalí a problémov nových technológií. Tieto problémy ma donútili k úprave návrhu a spomalili proces implementácie, nakoľko sa niektoré časti museli upraviť od základu. Implementácia prebiehala v iteráciách, pričom cieľom prvej iterácie bolo vytvorenie všetkých modulov aj napriek minimálnej funkcionalite. Výsledkom následných iterácií implementácie bol plne funkčný systém vo forme webovej aplikácie dostupnej na adrese www.contentu.tech.

Implementovaný systém prešiel programátorským a používateľským testovaním, ktorého sa zúčastnili štyri subjekty. Pred začiatkom testovania som jednotlivým subjektom predstavil systém a jeho funkcionalitu. Nasledovalo overenie funkcionality pomocou plnenia úloh z protokolu popísaného v časti 7.2. Všetky subjekty splnili úlohy bez problémov a časové rozdiely medzi nimi boli zanedbateľné. Poskytnutá spätná väzba bola zväčša pozitívna. Objavilo sa niekoľko pripomienok, ktoré kritizovali najmä návrh používateľského rozhrania.

8.1 Budúcnosť projektu

Pri návrhu tohto systému som mal veľa nápadov, na ktoré však nebol dostatok času. V rámci budúceho vývoja by som aplikáciu transformoval na PWA, ktorá by fungovala aj bez internetového pripojenia a po pripojení by automaticky synchronizovala dáta do databázy. Ďalším rozšírením by bolo pridanie tém a úpravy vzhľadu vytvorenej stránky. V súčasnosti systém očakáva, že každý tím má dostupný svoj vlastný server. Ďalším plánovaným rozšírením je úprava na centralizovaný systém, ktorý by podporoval viacero tímov naraz, a umožnil by aj kolaboráciu medzi sebou. Taktiež v systéme chýba možnosť integrácie s inými službami, ako napríklad sociálne siete.

Rád by som tento projekt zverejnil na platforme GitHub[1], vďaka čomu by sa na jeho vývoji mohli podieľať aj ostatní vývojári. Motiváciou tejto práce bolo poskytnutie služby, ktorá spríjemní a zefektívni workflow tvorcom obsahu. Za týmto si stojím, a preto by bola škoda, keby tento projekt skončil „v šuplíku“.

[1] GitHub

Literatúra

[1] Amazon. What is NoSQL? *AWS* [online]. 2020 [cit. 2020-4-1]. Dostupné z: https://aws.amazon.com/nosql/.

[2] Auth0. Introduction to JSON Web Tokens. *JSON Web Tokens* [online]. [cit. 2020-4-1]. Dostupné z: https://jwt.io/introduction/.

[3] Berners Lee, T., Fielding, R. T. a Masinter, L. *Uniform Resource Identifier (URI): Generic Syntax* [Internet Requests for Comments]. STD 66. RFC Editor, Január 2005. 8 s. http://www.rfc-editor.org/rfc/rfc3986.txt. Dostupné z: http://www.rfc-editor.org/rfc/rfc3986.txt.

[4] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*. Jún 1970, zv. 13, č. 6, s. 377–387. ISSN 0001-0782. Dostupné z: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf.

[5] Facebook. Introducing JSX. *Docs* [online]. 2020 [cit. 2020-2-25]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html.

[6] Gazarov, P. What is an API? In English, please. *#WEB DEVELOPMENT* [online]. December 2019 [cit. 2020-4-1]. Dostupné z: https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/.

[7] Google. MVC Architecture. *Google Chrome* [online]. 2020 [cit. 2020-2-25]. Dostupné z: https://developer.chrome.com/apps/app_frameworks.

[8] Kinsta. The Definitive PHP 5.6, 7.0, 7.1, 7.2, 7.3, and 7.4 Benchmarks (2020). *Kinsta Blog* [online]. 2020 [cit. 2020-2-24]. Dostupné z: https://kinsta.com/blog/php-benchmarks/.

[9] Martori, A., Bautista, B., Channell, J. a MacLeod, R. *2019 Website Threat Research Report* [online]. Výskumná správa. Sucuri, 2020 [cit. 2020-02-17]. Dostupné z: https://sucuri.net/wp-content/uploads/2020/01/20-sucuri-2019-hacked-report-1.pdf.

[10] MDN Contributors. What is the difference between webpage, website, web server, and search engine? *Mozilla* [online]. 2020 [cit. 2020-2-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Pages_sites_servers_and_search_engines.

[11] MDN Contributors. HTML: Hypertext Markup Language. *Web technology for developers* [online]. 2020 [cit. 2020-4-1]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.

[12] MDN Contributors. HTML5. *Web technology for developers* [online]. 2020 [cit. 2020-4-1]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5.

[13] Mohan, M. How React works under the hood. *#REACT* [online]. September 2019 [cit. 2020-4-1]. Dostupné z: https://www.freecodecamp.org/news/react-under-the-hood/.

[14] OpenJS Foundation. About Node.js. *Node.js* [online]. 2020 [cit. 2020-02-20]. Dostupné z: https://nodejs.org/en/about/.

[15] Oracle. What a Relational Database Is. *Oracle* [online]. [cit. 2020-2-24]. Dostupné z: https://www.oracle.com/database/what-is-a-relational-database/.

[16] Q-Success. Usage statistics of PHP for websites. *W3Techs* [online]. 2020 [cit. 2020-2-24]. Dostupné z: https://w3techs.com/technologies/details/pl-php.

[17] Q-Success. Usage statistics of content management systems. *W3Techs* [online]. 2020 [cit. 2020-2-17]. Dostupné z: https://w3techs.com/technologies/overview/content_management.

[18] Raymond, E. S. *The Art of Unix Programming*. Addison-Wesley Professional, 2003 [cit. 2020-2-17]. ISBN 0-13-142901-9.

[19] Red Hat. What is GraphQL? *API* [online]. [cit. 2020-4-1]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-graphql.

[20] Ryzac. What is REST? *Codecademy* [online]. [cit. 2020-4-1]. Dostupné z: https://www.codecademy.com/articles/what-is-rest.

[21] World Wide Web Foundation. History of the Web. *The Web* [online]. 2020 [cit. 2020-2-24]. Dostupné z: https://webfoundation.org/about/vision/history-of-the-web/.

Príloha A

Zoznam požiadaviek definovaných serverovou časťou

Názov	Argumenty	Návratový typ
getUser	id: ID!	User
getUsers		[User]!
getUserFromToken	token: String!	User
loginUser	email: String! password: String!	AuthData
getArticle	articleID: ID!	Article
getArticles	userID: ID	[Article]!
getBlogs	userID: ID password: String!	[Article]!
getTask	id: ID!	Task
getTasks	userID: ID	[Task]!
getPage	pageID: ID!	Page
getPages		[Page]!
getFiles		[String]
getConfig		Config

Tabuľka A.1: Zoznam požiadaviek definovaných v serverovej časti

Príloha B

Zoznam mutácií definovaných serverovou časťou

Názov	Argumenty	Návratový typ
createUser	email: String! trusted: Boolean bio: String role: String name: String	Boolean
changeUserPassword	userID: ID! password: String! currentPassword: String!	Boolean
updateUser	userID: ID! name: String bio: String trusted: Boolean role: String	Boolean
deleteUser	userID: ID!	Boolean
createArticle	title: String! content: String! isBlog: Boolean	Boolean
updateArticle	articleID: ID! title: String! content: String! isBlog: Boolean confirmed: Boolean published: Boolean	Boolean
deleteArticle	articleID: ID!	Boolean

Tabuľka B.1: Zoznam mutácií pre manipuláciu modelov User a Article

Názov	Argumenty	Návratový typ
createPage	title: String! description: String articles: [ID]	Boolean
updatePage	pageID: ID! title: String description: String articles: [ID]	Boolean
toggleArticleOnPage	pageID: ID! articleID: ID	Page
deletePage	pageID: ID!	Boolean
createTask	title: String! description: String assignee: ID! dueBy: String timeAllocated: Int	Boolean
updateTask	taskID: ID! title: String! description: String assignee: ID! dueBy: String timeAllocated: Int, timeSpent: Int	Boolean
deleteTask	taskID: ID!	Boolean
uploadFile	file: Upload!	Boolean
uploadProfilePicture	file: Upload!	String
deleteFile	filename: String!	Boolean
setConfig	siteName: String footerText: String	Boolean

Tabuľka B.2: Zoznam mutácií pre manipuláciu modelov Page, Task, File a Config

Príloha C

Obsah CD

Priložené CD obsahuje nasledovné:

- technicka_sprava – adresár s technickou správou a zdrojovými súbormi
- zdrojove_subory – adresár so zdrojovými súbormi aplikácie
- README – používateľský manuál