UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MODULÁRNY SYSTÉM PRE SPRAVOVANIE OBSAHU PREZENTÁCIE VÝSKUMNEJ SKUPINY NA INTERNETE

Bakalárska práca

2021 Ján Špirka

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MODULÁRNY SYSTÉM PRE SPRAVOVANIE OBSAHU PREZENTÁCIE VÝSKUMNEJ SKUPINY NA INTERNETE

Bakalárska práca

Študijný program: Aplikovaná informatika
Študijný odbor: Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava 2021 **Ján Špirka**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Ján Špirka
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Modulárny systém pre spravovanie obsahu prezentácie výskumnej skupiny na Internete
Modular system for content management of presentation of a research group on the Internet

Anotácia: Cieľom práce je navrhnúť, vyvinúť a nasadiť do prevádzky systém pre prezentovanie výskumnej skupiny na webe. Informácie o výskumnej skupine sú organizované v štruktúrovanej podobe, jednotlivé komponenty obsahu sa modulárne umiestňujú na webových stránkach. Administrátor môže nielen pridávať a spravovať obsah, ale aj definovať štruktúru stránok pomocou jednoduchých konfiguračných súborov. Systém umožňuje vkladať zoznamy a informácie o publikáciách, dokumentoch, projektoch, výučbe, študentských prácach, odkazoch, vytvárať nové články, zaraďovať videá. Od bežných CMS sa tento systém odlišuje tým, že má sadu predpripravených komponentov vhodných na vizuálne peknú, ľahko navigovateľnú, ale univerzálnu prezentáciu výskumnej skupiny. Systém bude vytvorený ako open-source a bude k dispozícii pre všetkých potenciálnych používateľov. Študent zároveň v spolupráci so školiteľom pripraví prezentáciu robotickej skupiny na Katedre aplikovanej informatiky. Systém bude vyvinutý využitím moderných webových technológií, ktoré študent predstaví v teoretickej časti práce.

Literatúra: WEBOVÁ PREZENTAČNÁ STRÁNKA ROBOTICKEJ SKUPINY, projekt TIS, https://github.com/SWED-team/TIS, 2016.
Tutorial: Intro to React, https://reactjs.org/tutorial/tutorial.html

Kľúčové slová: webová aplikácia, modularita, cms

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 21.01.2021

Dátum schválenia: 22.01.2021

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

....

študent vedúci práce

Čestné vyhlásenie

Čestne vyhlasujem, že bakalársku prácu som vypracoval samostatne s využitím teoretických a praktických vedomostí, odbornými radami školiteľa a s použitím uvedenej odbornej literatúry.

V Bratislave dňa 21.5.2021 .....

Ján ŠPIRKA

Poďakovanie

Ďakujem vedúcemu mojej bakalárskej práce Mgr. Pavlovi Petrovičovi, PhD. za pomoc a odborné vedenie, ktoré mi poskytol pri vypracovávaní bakalárskej práce. Rovnako ďakujem priateľke a rodine za ich podporu.

ABSTRAKT

Content management system alebo aj CMS si získali značné množstvo používateľov vďaka svojej jednoduchosti a množstvu nástrojov, ktoré ponúkajú. V tejto práci objasňujeme základné pojmy týkajúce sa problematiky CMS. Venujeme sa analýze existujúceho riešenia, ktoré mala doposiaľ k dispozícii výskumná skupina venujúca sa robotike. Zvažujeme výber vhodných technológií potrebných na implementáciu nového, viac modulárneho riešenia. Následne sa v praktickej časti venujeme samotnej implementácii webovej aplikácie na základe požiadaviek robotickej skupiny, grafického návrhu a návrhu databázového modelu. Výsledkom práce je univerzálny open-source systém na správu modulárneho webového obsahu určený primárne na prezentáciu ľubovoľnej výskumnej skupiny. Systém bol úspešne nasadený do prevádzky pre výskumnú skupinu na Katedre aplikovanej informatiky FMFI UK a pripravuje sa prezentácia ďalšej výskumnej skupiny na rovnocennej katedre.

Kľúčové slová: webová aplikácia, modularita, CMS

ABSTRACT

The content management system or CMS has gained a large number of users thanks to it’s simplicity and the number of tools they provide. In this theis, we clarify the basic concepts related to CMS problematics. We analyze the existing solution, which was available to a scientific group dedicated to robotics. We are considering the selection of suitable technologies needed to implement a new, much more modular solution. Subsequently, in the practical part, we focus on the implementation of the web application based on the requirements of the robotics group, graphic design, database model design. The result of the work is an open-source system for the management of modular web content designed primarily for the presentation of any research group. The system was successfully put into operation for a research group at the Department of Applied Informatics FMFI UK and a presentation of another research group is being prepared.

Keywords: web aplication, modularity, cms

Obsah

Úvod

V súčasnosti je potrebné mať všetky dostupné informácie na webe. Tieto informácie alebo aj obsah stránok sa však väčšinou neustále mení a je treba ho udržiavať a vylepšovať aby bol vždy aktuálny. Zmeny v tradičných webových prezentáciách vyžadovali zásah programátorov, ktorí museli zmeniť kód, upraviť dáta v databáze a každú novú funkcionalitu museli ručne programovať, čo značne zvyšovalo náklady na chod a údržbu stránok.

Robotická skupina pri Katedre aplikovanej informatiky má už dlhodobo potrebu prezentovať svoju prácu, propagovať výsledky svojej práce alebo priamo poskytovať výučbové informácie prostredníctvom webu. Tradičné riešenie by pri veľmi dynamickom obsahu stránok spôsobilo veľa starostí spojených s vývojom a údržbou, preto je vhodné nasadenie nejakého CMS systému. Univerzálne CMS systémy dovoľujú vytvoriť na webe kvalitnú a takmer ľubovoľnú prezentáciu. Bohatosť ich funkcionality však vedie k tomu, že ich spravovanie si dnes vyžaduje skúseného správcu, ktorý okrem pravidelných aktualizácií, orientovania sa v zložitých paneloch a menu systémoch aj tak musí vytvárať špecifické šablóny a prispôsobovať vzhľad. Pri všetkom takomto úsilí výsledok stále nemusí poskytovať želanú funkcionalitu v takej podobe ako by autori obsahu potrebovali. Ideálnym sa preto javil vývoj vlastného CMS, vyladeného presne pre danú aplikáciu, ktorý z dlhodobého hľadiska značne pomáha a zjednodušuje prácu.

V práci sa snažíme čitateľovi objasniť niektoré pojmy, s ktorými sa môže pri problematike CMS stretnúť a v jednotlivých kapitolách popísať postup, ktorý sme si zvolili pri tvorbe nášho vlastného CMS.

Prvá kapitola sa primárne venuje problému, ktorý sme sa snažili riešiť a prvotnému riešeniu, ktoré vzniklo pred niekoľkými rokmi. Preberáme v nej architektúru predchádzajúceho riešenia, jeho výhody aj nevýhody. Kapitolu uzatvárame zhrnutím toho, čo by bolo možné zo starého riešenia použiť.

V druhej kapitole opisujeme CMS ako také a analyzujeme vybrané existujúce CMS riešenia, čo nám pomohlo pri výbere funkcionalít, ktoré sme sa rozhodli naimplementovať v našom riešení.

Treťou a štvrtou kapitolou sa končí teoretická časť našej práce a začína tá praktická. Nazbierané poznatky bolo treba zjednotiť a vytvoriť grafický návrh, ktorý nám pomohol pri implementácii. Tiež bolo treba vytvoriť architektonický návrh a na jeho základe vybrať vhodné technológie na implementáciu nášho riešenia.

V piatej kapitole sa snažíme opísať proces implementácie riešenia spolu s problémami, na ktoré sme narazili a v poslednej kapitole prezentujeme výsledky práce.

1 Prehľad problematiky

1.1 Predchádzajúce riešenie

Potreba samostatnej webovej prezentácie tvorby vlastného CMS vznikla už skôr vzhľadom na rozvoj robotiky na našej katedre. Bolo potrebné vytvoriť jednoduchý nástroj, s prívetivým dizajnom a s funkcionalitou dostatočnou na vytvorenie reprezentatívne vyzerajúceho webu.

Predchádzajúce riešenie je z roku 2015, kedy na ňom pracovala štvorčlenná skupina študentov v rámci predmetu Tvorba informačných systémov na našej fakulte. Neskôr ho vylepšila jedna študentka ako projekt na predmet Životný cyklus informačných systémov. Funkcie výsledného riešenia sa delia podľa troch rolí používateľov, ktorými sú administrátor, registrovaný používateľ a návštevník. Administrátor môže pridávať, editovať a mazať obsah vo všetkých stránkach v aplikácii. Zároveň schvaľuje zmeny vykonané registrovanými používateľmi. Tí môžu spravovať obsah stránok, ktoré vyrobili alebo im bol pridelený súhlas na editovanie. Návštevník môže prezerať všetky zverejnené príspevky a má možnosť registrácie, po ktorej sa stáva tvorcom obsahu. Registrácia takisto podlieha schváleniu administrátora.

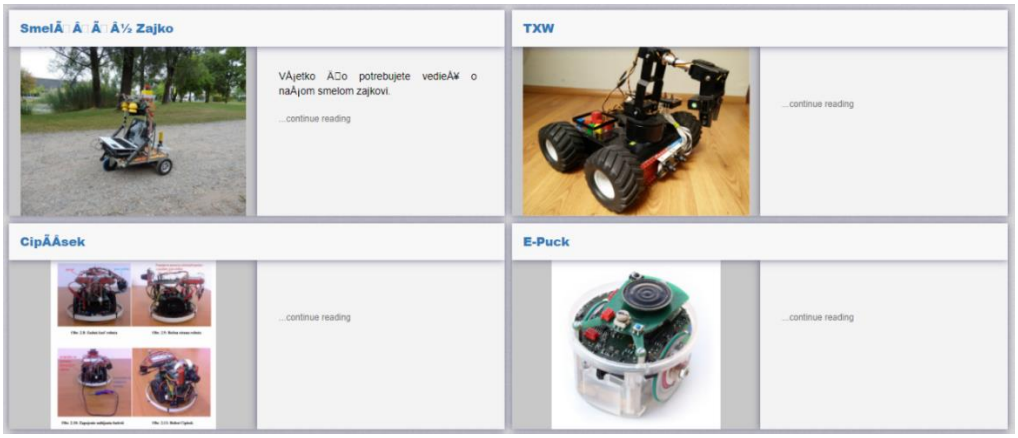
1.1.1 Návrh a architektúra

Systém je postavený na tradičnom návrhovom vzore *Model-view-controller* (MVC), ktorý pomáha k lepšej organizácii a teda aj udržateľnosti kódu. Ako názov prezrádza, skladá sa z troch zložiek:

- *Model*, zodpovedá všetkej logike týkajúcej sa údajov, s ktorou používateľ pracuje. To môže predstavovať buď údaje, ktoré sa prenášajú medzi časťami *View* a *Controller*, alebo pri manipulácii priamo s databázou.
- *View* sa používa pri používateľskom rozhraní aplikácie na zobrazenie všetkých potrebných dát a ich manipulácii. Napríklad zobrazenie stránky bude obsahovať všetky komponenty používateľského rozhrania, s ktorými koncový používateľ interaguje.

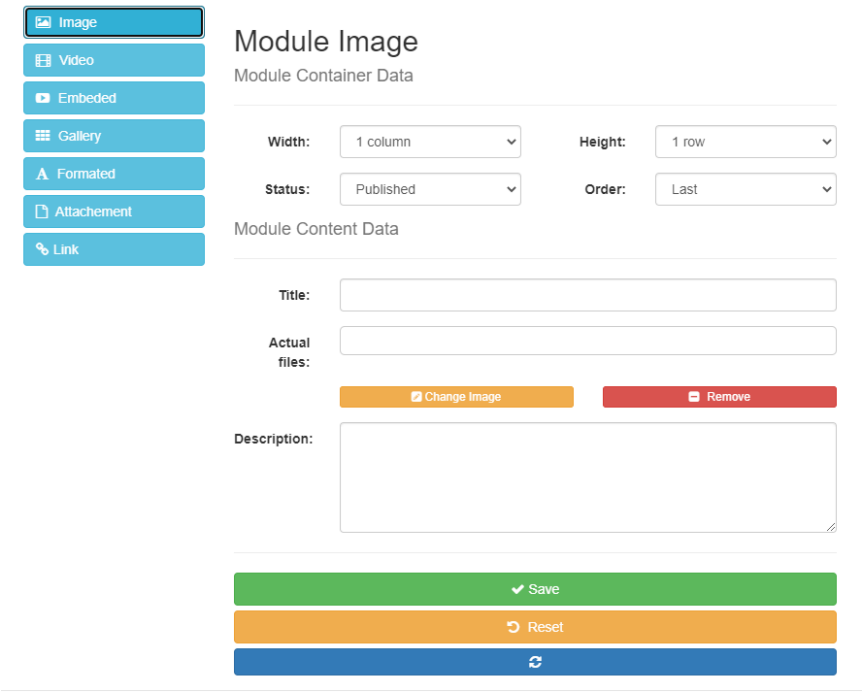
- *Controller* funguje ako rozhranie medzi časťami *Model* a *View*, aby spracovali všetky prichádzajúce požiadavky, manipulovali s dátami pomocou časti *Model* a interagovali s *View* na vykreslenie konečného výstupu. [1] [2] [3]

Návrh používateľského rozhrania zobrazujúci články bol inšpirovaný podľa oficiálnej stránky NASA, ktorá svoj bohatý obsah člení do viacerých kategórií v navigačných paneloch a náhľadoch. Náhľady sú zobrazené v mriežke a po kliknutí na jeden z nich sa zobrazí ich detailný obsah, ktorým je väčšinou článok, galéria alebo ďalšia mriežka z ktorej si používateľ opäť môže vyberať. Na obrázku 1 vidno ako sa študentom podarilo vytvoriť jej napodobneninu.



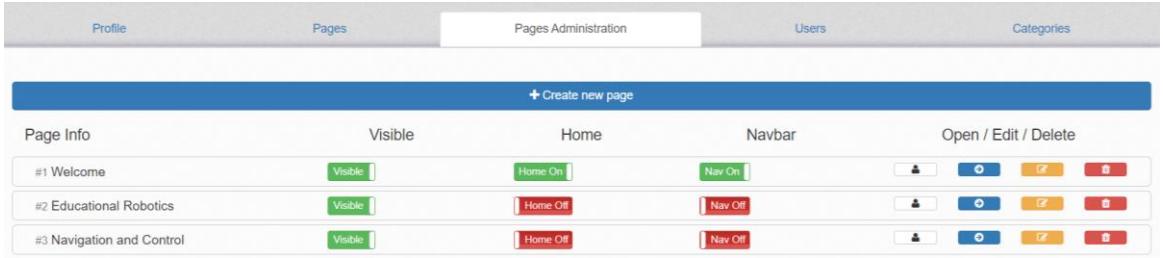
Obrázok 1: Náhľady v mriežke, študentské riešenie, Zdroj: [4]

Vytvorenie obsahu na novej stránke spočíva v pridávaní modulov. Rozloženie stránky je aj v tomto prípade robené formou mriežky a používateľ môže nastaviť rozmery a pozíciu modulov. K dispozícii je šesť typov, galéria, obrázok, video, formátovaný text, hypertextový odkaz a príloha súboru. Tieto moduly sú na stránke zobrazené v poradí v akom ich používateľ zadá, ale dá sa aj dodatočne meniť. Každá stránka sa môže skladať z ľubovoľného počtu takýchto modulov. Ako vidno na obrázku 2, pri vytváraní nového modulu sa otvorí formulár, v ktorom je potrebné nastaviť šírku, výšku, status (viditeľnosť) a jeho poradie na stránke. Okrem toho používateľ musí novému modulu nastaviť nadpis a krátky popis, prípadne priložiť súbory, ktoré bude modul obsahovať.



Obrázok 2: Modal na vytváranie nového modulu, Zdroj: [4]

Nasledujúci obrázok 3 obsahuje ukážku tabuľky všetkých stránok, ktoré sú v systéme vytvorené. Administrátor môže každej stránke nastaviť či je pre návštevníka viditeľná a či sa odkaz na danú stránku bude vyskytovať v navigácii. Zároveň môže vybrať, ktorá zo stránok bude označená ako domovská. Každá stránka z tohto zoznamu obsahuje tlačidlo pre jej otvorenie, editáciu alebo zmazanie. Ak chce používateľ vytvoriť novú stránku, tak to môže urobiť jedine na tomto mieste stlačením tlačidla *"Create new page"*.



Obrázok 3: Administrácia stránok, Zdroj: [4]

1.1.2 Použité technológie

Na implementáciu predchádzajúceho riešenia boli použité technológie HTML, CSS, PHP a JavaScript. K jazyku PHP nebol použitý žiadny ďalší *framework*. Všetky údaje mala

na starosti databáza MySQL a prácu s front-endom uľahčovala knižnica Bootstrap a jQuery. jQuery bola v tom čase najpoužívanejšia knižnica na manipuláciu s DOM a PHP bol zas najpopulárnejší *server-side* programovací jazyk. [5]

1.1.3 Nedostatky riešenia

Webová aplikácia bola budovaná len krátku dobu, preto z jej výsledku vzišlo viacero nedostatkov. S pomedzi tých dôležitejších možno vybrať, že aplikácia ponúka na výber šesť modulov, ktoré majú len veľmi malú možnosť nastavenia. Prakticky sa dá pri každom module okrem jeho umiestnenia nastaviť už len nadpis, popis a prípadná príloha.

Pri snahe zmeniť poradie, v ktorom sa má modul zobraziť na stránke, je potrebné dostať sa do jeho nastavení a vybrať tam jeho poradové číslo. V prípade, že je na stránke veľa modulov, používateľ si musí dobre spočítať, na ktoré poradie ho chce uložiť. Zmena poradia zároveň ľahko rozhodí poradie ostatných modulov a dostať stránku do predošlého kroku je v niektorých prípadoch veľmi zdĺhavé.

Pri vytváraní novej stránky používateľ nedokáže ovplyvniť to, ako bude vyzerať jej URL adresa. Tá je totiž v pevnom formáte */page=id*, kde id je vygenerovaný identifikátor stránky. Okrem toho chýba možnosť ako nastaviť poradie odkazov v navigácii a ich zoskupovanie, respektíve možnosť vytvoriť v navigácii *dropdown*. V navigačnom paneli je umiestnené vyhľadávanie, ktoré ale takmer nikdy nenájde hľadaný výraz.

Aby bolo vidno hocijakú vykonanú zmenu, používateľ musí manuálne obnoviť stránku, aby videl jej zmenený obsah, čím aplikácia stráca na príjemnej používateľskej skúsenosti pri editovaní stránky. Navyše má aplikácia problém so zobrazovaním mäkčeňov a podobných špeciálnych znakov.

1.1.4 Využiteľnosť riešenia

Výsledkom je síce stránka nasadená na školskom serveri, avšak odkaz na ňu nie je nikde zverejnený. Neplní viaceré základné požiadavky dostatočne na to, aby mohla prebrať svoju reprezentatívnu funkciu. Systém už pôsobí zastaralo a nie je dostatočne responzívny pre zariadenia menšie ako počítač. Napriek tomu jej vypracovanie pomohlo pri návrhu a pri predstave používateľského rozhrania a zároveň nám poskytlo skúsenosti s používaním

vlastnej verzie CMS systému, ktorým sa budem venovať v ďalšej kapitole. V našom návrhu zachovávame koncept administrácie stránok a používateľov vo forme tabuliek, vytváranie obsahu pomocou modulov rôznych typov uložených v mriežke a aj role používateľov.

2 Prehľad podobných systémov

V dnešnej dobe je veľký dopyt po webových stránkach, ktoré umožňujú menším podnikateľom rásť alebo bežným ľuďom prezentovať svoje názory, skúsenosti či zážitky. Takíto ľudia nepotrebujú vždy systém vytvorený na mieru a preto existuje veľké množstvo systémov, ktoré slúžia na zjednodušenie tvorby webových stránok až do takej miery, že to zvládne aj laik. Poznáme ich pod spoločným názvom *Content management system* (CMS), ktorý by sa do slovenčiny dal preložiť ako systém na spravovanie obsahu. V kapitole uvedieme základnú charakteristiku týchto systémov a spomenieme štyri produkty, ktoré patria k najpoužívanejším v tejto oblasti.

2.1 CMS

Zjednodušenie tvorby webových stránok je dosiahnuté viacerými vlastnosťami samotného systému. Samozrejme jednotlivé produkty sa snažia dosiahnuť želaný výsledok svojím vlastným spôsobom a tým sa odlíšiť na trhu, ale vo všeobecnosti by sa dalo generalizovať niekoľko pravidiel, alebo skôr niekoľko potrieb klienta, ktoré riešia všetky systémy.

Princípom fungovania je, že CMS vytvára vrstvu nad všetkou prácou, ktorú by inak musel urobiť programátor. Je to súbor nástrojov a funkcií, ktoré umožňujú oveľa ľahšie vytvárať a aktualizovať obsah bez toho, aby sa niečo muselo ručne programovať. Používajú sa už dopredu pripravené komponenty, ktoré slúžia ako šablóna. [6]

Obsah, ktorý je zobrazovaný na stránke sa dá editovať priamo v prehliadači a to väčšinou bez potreby zásahu do zdrojových súborov. Tým sa ľubovoľný prehliadač stáva vývojovým prostredím pre klienta, ktorý si môže upravovať obsah podľa svojich potrieb a nemusí pri tom využívať pomoc programátora, ktorý by obvykle zmenu musel dosiahnuť v zdrojovom kóde. Paleta nástrojov, ktorá je ponúknutá klientovi je obmedzená a teda v prípadoch, kedy by úprava obsahu prekračovala tieto možnosti, neostáva nič iné ako požiadať poskytovateľa služby o doplnenie novej funkcionality do palety nástrojov, ktorá by mohla byť neskôr dostupná aj ostatným klientom, alebo formou riešenia na mieru v zdrojových súboroch toho konkrétneho projektu.

Ako vidíme, tak aj klient s nízkou, či dokonca žiadnou znalosťou programovacích alebo značkovacích jazykov dostáva možnosť relatívne ľahko vyskladať obsah stránky podľa svojich predstáv a následne ho upravovaním udržiavať aktuálny. [7]

Stále však platí, že dobrá stránka potrebuje mať dobrý dizajnový návrh. Ten sa dá do značnej miery nahradiť ponúkanými šablónami, avšak potom sa stáva, že sa stránky až príliš podobajú a stráca sa ich jedinečnosť čím strácajú pútavosť a znižuje sa šanca, že sa návštevník na takúto stránku ešte v budúcnosti vráti. Na stránky, ktoré často menia svoj obsah a nepotrebujú si budovať značku je to však stále dobrá a hlavne lacnejšia voľba.

Webové CMS zvyčajne klientovi umožňujú:

- spravovanie obsahu
- spravovanie súborov a dokumentov
- plány webhostingu
- spravovanie používateľov
- interakcie s ďalšími webovými systémami

Cena je dôležitým faktorom, ktorý hrá v prospech použitia CMS. Hlavne pri tradičných weboch, ktoré je potrebné neustále upravovať sa môžu náklady na pracovnú silu nazbierať a značne predražiť prevádzkovanie stránky. V niektorých prípadoch je používateľ úplne schopný nahradiť programátora v procese tvorby a údržby webu, čo dokáže značne zredukovať náklady. Medzi najznámejšie CMS patria štyri nasledujúce systémy.

2.1.1 Wordpress

Wordpress je jednoznačne jednotkou na trhu a funguje už od roku 2003. Za tú dobu získal dominantnú pozíciu na trhu a svoje služby poskytuje aj zadarmo. Jeho veľkou výhodou je, že je *open source* a je možné mať vlastnú inštanciu, nad ktorou by sme mali plnú kontrolu. Je treba rozlišovať medzi wordpress.org a wordpress.com. Wordpress.org je organizácia, ktorá má na starosti prevádzkovanie wordpress.com. Wordpress.com je už priamo softvér, ktorý môže zadarmo využiť koncový používateľ za určitých podmienok.

Tento softvér umožňuje mnoho funkcionalít napr. používanie tém a hotových *template-ov* stránok alebo rôzne pluginy slúžiace na optimalizáciu či obohatenie

funkcionality výslednej stránky. Služba poskytuje aj *hosting*, avšak nie pod vlastnou doménou. Tá bude vždy v tvare „meno.wordpress.com“. Vlastnú doménu môže používateľ použiť až vtedy, keď prevádzkuje svoju vlastnú inštanciu Wordpressu.

2.1.2 Wix

Moderný systém, ktorý umožňuje vytvárať a upravovať UI komponenty metódou *drag and drop*. Medzi tisíckami inovatívnych funkcií patrí množstvo animácií, ktoré oživia dizajn, optimalizácia pre mobilné telefóny a iné zariadenia, SEO, záruka bezpečnosti, či analýza návštevnosti a podobne. To že je produkt zameraný predovšetkým na front-end dokazuje aj *Wix ADI*. Nástroj, ktorý si od používateľa vyžiada odpovede na niekoľko otázok, na základe ktorých mu navrhne návrhy na dizajn, ktoré môže ďalej slobodne upravovať. Spoločnosť vyvinula aj pomocný editor, ktorý je spustiteľný priamo v prehliadači a ešte viac zjednodušuje prácu s dizajnom.

Wix okrem vytvorenia samotnej stránky pomáha aj s vytvorením a tlačou loga. Po veľkej marketingovej investícii zaznamenáva stúpajúci trend a stáva sa čoraz viac používaným produktom po celom svete. [8]

2.1.3 Sanity

Moderná platforma určená pre web developerov, používajúca JavaScript a vlastnú technológiu GROQ, ktorá je veľmi podobná GraphQL, avšak na rozdiel od nej nepotrebuje definovať *rezolvery* ani schémy. Ponúka viaceré vstavané komponenty, ktoré sú voľne programovateľné, čo ďalej rozširuje funkcionalitu. Ďalšou veľkou výhodou je podpora spolupráce viacerých členov tímu, ktorí vidia zmeny v reálnom čase a pokročilý *version control*. Podľa ich oficiálnej stránky sa jedná o jediný CMS, ktorý ponúka tieto možnosti na tak vysokej úrovni.

Okrem toho je radené medzi tzv. *headless* CMS, čo znamená, že ako front-end sa môže použiť hocijaký *framework*. V porovnaní so systémom Wix teda uľahčuje riadenie obsahu viac zo strany *backend-u* než *frontend-u*. V tomto prípade by klient bez znalosti programovania bez podpory nezvládol vytvoriť vlastnú stránku. Sanity je vytvorený na uľahčenie práce predovšetkým pre developerov.

2.1.4 MediaWiki

Štvrtý výber produktu je takisto odlišný, pretože sa nejedná o klasický moderný CMS. Je prispôsobený pre potreby zverejňovania článkov, respektíve Wikipedia projektov, ktoré nesú hlavne informačnú hodnotu. Registrovaný používateľ môže vytvoriť alebo upraviť článok vkladaním formátovaného textu, obrázkov, tabuliek, grafov a veľa ďalších stiahnuteľných rozšírení, ktoré sú programované v jazyku PHP.

MediaWiki si získala popularitu najmä vďaka tomu, že je primárne určená pre webové prezentácie s obsahom, na ktorého tvorbe spolupracuje množstvo autorov. Preto dôsledne udržuje záznamy o všetkých verziách stránok, jednotlivé čiastkové úpravy je možné vizualizovať, porovnávať rozličné verzie, odstrániť, diskutovať o obsahu a podobne. Definuje vlastný jazyk na popis obsahu, ktorý je prístupnejší a hlavne čitateľnejší ako HTML a používatelia sa ho vedia rýchlo naučiť používať. Podobne ako pre iné CMS platí, že existujú desiatky rozšírení – modulov s rozličnou funkcionalitou, ktorými je možné webovú prezentáciu obohatiť. Mechanizmus vytvárania vlastných rozšírení je natoľko jednoduchý, že ho zvládnu aj začiatočníci.

Aj keď sa to bežnému návštevníkovi Wikipédie možno nezdá, MediaWiki sa snaží napredovať a používať nové technológie. Napríklad v roku 2020 sa začalo experimentovať s návrhom prispôsobiť moderný Javascript *framework* Vue.js s MediaWiki. Po roku testovania sa využitím niektorých jeho vlastností podarilo rozšíriť svoju funkcionalitu. [9]

2.2 Výstup analýzy

Všetky z vyššie spomenutých CMS by dokázali zabezpečiť určitú časť funkcionality našej webovej aplikácie. MediaWiki sa najmenej podobá finálnemu riešeniu, ktoré sa snažíme docieliť, ale možnosť vytvárať články pomocou jednoduchého nástroja sme zaradili do výslednej funkcionality. Nástroj Sanity je určite zaujímavý, pretože umožňuje použitie vlastného *frontend-u*, čo je v mnohých prípadoch užitočné, ale v našom prípade je jednoduchšie použiť vlastný *backend*, ktorý by mohol byť spustený na školských serveroch a pracoval by s nami zvolenou databázou. Keď ide o Wix a Wordpress, je ťažké povedať, ktorý z nich je lepší. Obidva poskytujú množstvo funkcií, ktoré presahujú naše potreby, čo

je možno najväčšou nevýhodou týchto systémov, lebo sa v nich môže používateľ ľahko stratiť.

Rozhodli sme sa preto vytvoriť CMS, ktoré bude hlavne prehľadné, intuitívne, úzko zamerané pre náš prípad použitia a čo najviac odbremení používateľa od ťažkostí s vytváraním grafického rozhrania a prezentácie.

3 Návrh a architektúra

Webová aplikácia je typickým príkladom architektúry klient – server. Klient je žiadateľ služby a server je jej poskytovateľ. Cez webový prehliadač si klient vyžiada od webového servera obsah stránky, ktorú po jeho prijatí zobrazí. Spojenie medzi nimi spravidla začína klient a prebieha cez počítačovú sieť na základe komunikačného protokolu.

Tradičný spôsob, s ktorými fungujú technológie ako PHP alebo WordPress je *server side rendering* (SSR), pomocou ktorého sa výsledné html pripravuje na serveri. Používateľ pre načítanie stránky, pošle požiadavku na server, ktorý skontroluje jej zdroj, spracuje a pripraví HTML súbor. Pošle ho naspäť do prehliadača, ktorý si ho stiahne a zobrazí jeho obsah. Prehliadač si ešte stiahne a spustí Javascript súbor, ktorý ďalej slúži na interakciu používateľa so stránkou. Najväčšou nevýhodou takéhoto prístupu je, že pri prechode na ďalšiu stránku sa celý proces opakuje od začiatku, čím sa okrem *cachingu* neušetrí žiadny čas.

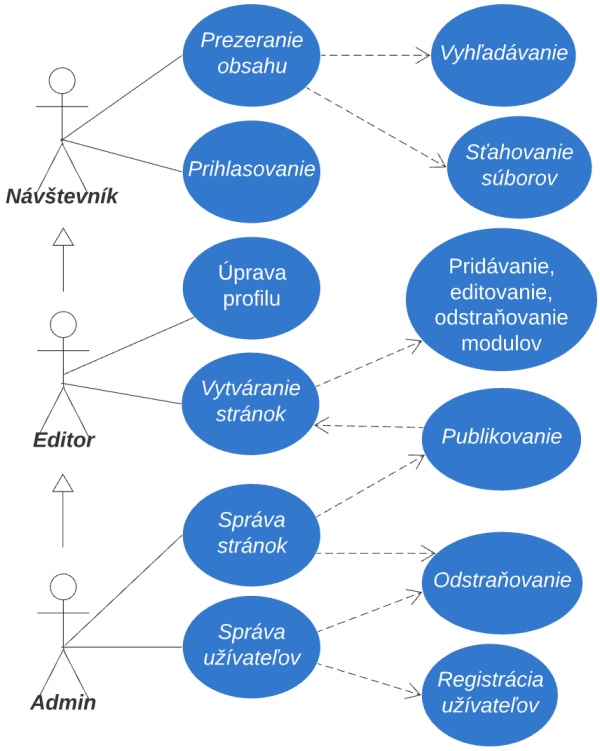
Na druhej strane *client side rendering* (CSR) vytvára obsah na strane klienta, čo umožňuje, aby sa webová stránka prekreslila v prehľadávači pri prechode na rôzne stránky. Pri prvom načítaní stránky si zo servera stiahne HTML súbor, ktorý ale obsahuje oveľa menej informácii ako to bolo v predošlom príklade a následne Javascript súbor, ktorý ďalej manažuje zvyšok práce na strane klienta. Ten má od toho momentu na starosti vykresľovanie všetkých komponentov, z ktorých sa skladá celá stránka. Javascript komunikuje so serverom už len v prípadoch, kedy potrebuje poslať požiadavku na získanie alebo zapísanie nových dát do databázy.

Hlavný rozdiel je v dobe načítania webovej stránky, teda času, ktorý uplynie medzi odoslaním žiadosti na server a jej vykreslením v prehliadači. Je to dôležitý aspekt, pokiaľ ide o dojem, ktorý získava používateľ. Prvé načítanie stránky pochopiteľne trvá o trochu kratšie v prípade SSR, pretože mu stačí balík dát potrebný na zobrazenie len jednej konkrétnej stránky. Výhodu ale preberá CSR v scenároch, kedy sa používateľ často preklikáva medzi jednotlivými stránkami aplikácie. Nakoľko si už všetko riadi Javascript a nepotrebuje nutne spojenie so serverom, tak aplikácia čiastočne funguje aj v off-line režime, pretože viaceré veci potrebujúce na svoj chod už má stiahnuté. Samozrejme stránka

už nebude fungovať podľa predstáv v prípadoch, kedy klient posiela požiadavku na server alebo potrebuje stiahnuť CSS či Javascript súbor z internetu. Off-line režim by sa dal ešte o niečo vylepšiť využitím *service workers* a dátového úložiska prehliadača. [10] [11]

CSR je v konečnom dôsledku lepšie na webové aplikácie, pretože tie zvyčajne zahrňujú väčšiu interakciu s používateľom. Pocit z rýchleho prechádzania medzi stránkami poteší každého používateľa a tak stojí za to pri prvom načítavaní stránky počkať trošku dlhší čas. [12]

3.1 Používateľské role



Obrázok 4: Use-case diagram zobrazujúci role a funkcie používateľov

Obrázok 4 znázorňuje, že aplikácia je rozdelená medzi tri role používateľov. Návštevník, je neprihlásený používateľ, ktorý môže prezerať obsah stránky alebo sťahovať priložené súbory. V prípade, že má návštevník vytvorené konto, tak sa môže prihlásiť. V tom okamihu prechádza do role editora, kedy už môže vykonávať zmeny. Prihlásený používateľ si môže zmeniť profilové údaje, ale čo je dôležitejšie, môže meniť samotný obsah stránok. Má však určité obmedzenia. Môže upravovať len stránky, ktoré sám vytvoril.

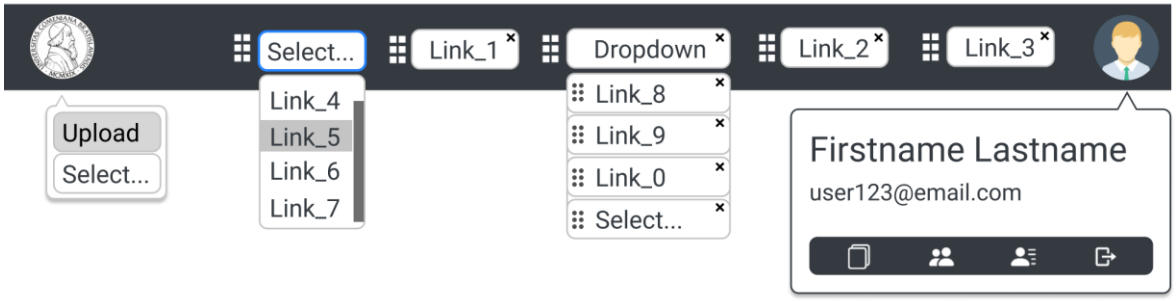
Navyše na ich publikovanie potrebuje schválenie administrátora. Administrátor okrem spravovania všetkých stránok má na starosti aj správu používateľov a navigácie.

3.2 Návrh rozhrania

Pri navrhovaní používateľského rozhrania nám ako dobrý základ pomohlo pôvodné riešenie. Základný koncept administrácie stránok a používateľov ostal okrem niekoľko zjednodušení prakticky bez zmeny. Hlavné časti aplikácie, ktoré potrebovali graficky vylepšiť boli navigácia a stránka, na ktorej sa dá vytvárať obsah (*custom page*).

Pre lepšiu predstavu pred implementáciou sme na grafický návrh použili webový nástroj Figma. Je to editor vektorovej grafiky a prototypový nástroj používateľského rozhrania. Sada funkcií spoločnosti Figma sa zameriava na použitie s dôrazom na spoluprácu v reálnom čase. Tým sa najviac líši od podobných nástrojov akými sú napríklad Sketch alebo Adobe XD. Ďalšou veľkou výhodou je to, že Figma dokáže na základe návrhu vygenerovať CSS, ktoré je možné použiť pri programovaní našich komponentov, čím nám značne urýchli vývoj. [13]

Na nasledujúcom obrázku 5 je výsledok dizajnového návrhu pre navigačný panel, ktorý je ladený s CSS, ktorý používa Bootstrap. Zľava doprava je na ňom vidieť logo aplikácie, odkazy na vytvorené stránky, ich zoskupenie (*dropdown*) a posledná ikona predstavuje fotku používateľa s modalom, ktorý obsahuje tlačidlá akcií.

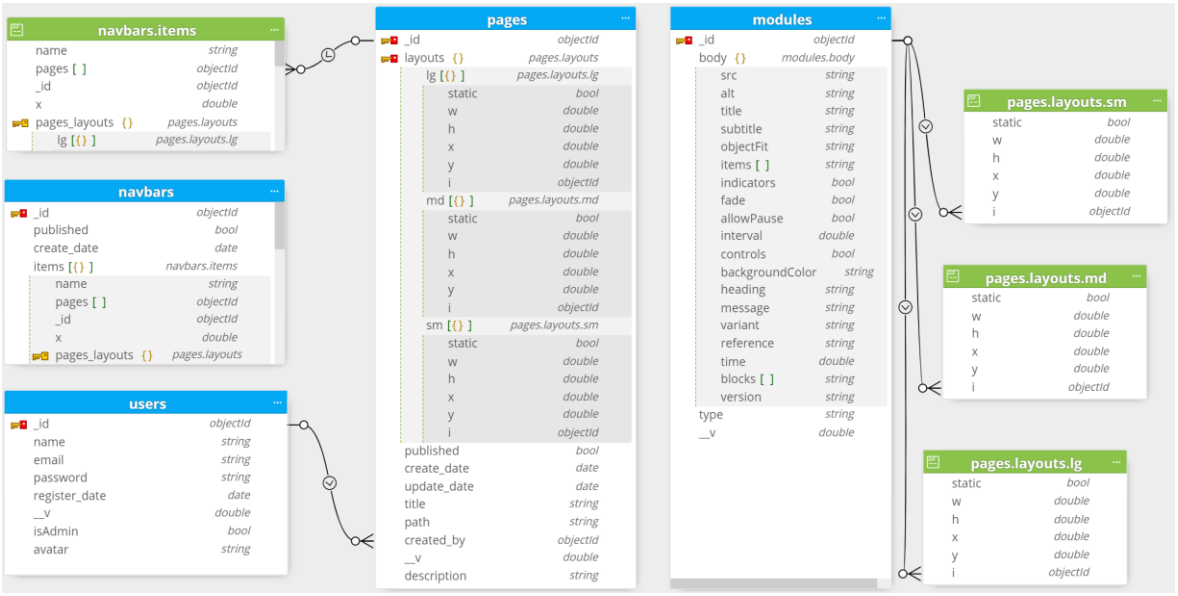


Obrázok 5: Návrh navigačného panelu s možnosťami nastavenia v nástroji Figma

Návrh na *custom page* bol opäť inšpirovaný stránkou NASA, čo znamená, že jednotlivé moduly sú štvoruholníkové útvary rozložené v mriežke v rôznych veľkostiach. Ak je používateľ práve prihlásený a má povolenie upravovať stránku, tak sa po prejdení myšou

ponad modul zobrazí malý panelový nástroj priamo nad ním a po kliknutí na modul sa zobrazí bočný panel, ktorý obsahuje formulár, ktorého editovanie hneď mení aj obsah aktívneho modulu.

3.3 Kolekcie v databáze



Obrázok 6: Model databázy

V rámci databázy sa vyskytujú kolekcie *pages*, *modules*, *users* a *navbars*. Na obrázku 6 sú čiarami vyznačené vzťahy medzi jednotlivými kolekciami.

Každý stránka obsahuje základné informácie ako názov, cesta (URL adresa), id autora, dátum vytvorenia či poslednej aktualizácie. Ku každej stránke je možné pridať aj krátky popis, ktorý pre návštevníka stránky nie je nikde viditeľný, slúži skôr ako poznámka pre registrovaných používateľov. Stránky je tvorená z troch rozložení: *sm*, *md*, *lg*. Vždy sa zobrazuje práve jedno na základe šírky obrazovky. Povinné je len najväčšie z nich a teda *lg* rozloženie. Všetky moduly sú v stránke usporiadané v jednotlivých rozloženiach podľa osi x, y a podľa svojej veľkosti. Moduly môžu mať v každom rozložení odlišné usporiadanie. Novovytvorená stránka sa do databázy uloží s hodnotou *false* pre kľúč *published*, čo znamená, že nie je viditeľná pre odhláseného používateľa. Túto hodnotu môže zmeniť jedine administrátor.

Každý modul má jednoznačný *type* potrebný na rozlíšenie komponentov pri renderovaní. Hodnota kľúča *body* je závislá takisto od vybraného typu. Pri vytvorení modulu typu *image* sa dá definovať jeho zdroj, zarovnanie, popis a iné, ale napríklad pre typ *alert* len farba a text. Niektoré vybrané typy majú kľúč *reference*, ktorým sa odkazujú na internú alebo externú stránku, na ktorú sa návštevník dostane po kliknutí na modul.

Každá stránka sa pod svojim názvom môže zobraziť v navigačnom paneli. Aplikácia podporuje jednoduchý *versioning* pre navigačný panel, preto kolekcia *navbars* obsahuje jedinečné id, dátum vytvorenia a informáciu o tom, či je práve publikovaná. Backend zabezpečí, že hodnota *true* sa vyskytuje v práve jednom dokumente. Samotný obsah navigačného panelu je uložený pod kľúčom *items*, ktorého hodnota je typu pole objektov a každý objekt obsahuje svoju pozíciu na osi x. Pole *pages* obsahujúce referencie na existujúce stránky z kolekcie stránok. V prípade, že toto pole obsahuje viac ako jeden prvok, tak je potrebné uložiť ich spoločný názov ako hodnotu do kľúča *name*.

Každý používateľ má vo svojom profile informácie o mene, emaily, hesle a dátumu registrácie. Hodnota pre kľuč *isAdmin* hovorí o tom, či je daný používateľ administrátorom alebo nie.

4 Použité technológie

V tejto kapitole podávame informácie o hlavných technológiách, ktoré sme vybrali na implementáciu zadania. Technológie z predošlého riešenia porovnávame s našim modernejším výberom, ktorý ich nahradil.

4.1 Frontend

Na strane klienta sme použili čoraz viac populárny *framework* React, ktorý je vďaka svojim vlastnostiam vhodný na vypracovanie riešenia. Dve kľúčové knižnice, ktoré nám pri dosiahnutí cieľových požiadaviek pomohli boli React-grid-layout a Editor.js. Obidve majú relatívne malú veľkosť balíčka a potrebujú len dve ďalšie závislé knižnice.

4.1.1 React

React sa stal veľmi populárnym a môžeme povedať, že už nahradil jQuery. Zatiaľ čo jQuery pracuje s HTML DOM priamo, tak React používa virtuálny DOM, vďaka čomu je oveľa rýchlejší. Rozdiel medzi bežným a virtuálnym DOM sa ukáže, keď je v ňom potrebné niečo aktualizovať. Aj keď tento proces nie je taký zložitý, veci sa môžu začať spomaľovať, ak sa DOM potrebuje aktualizovať opakovane, alebo ak sa strom čoraz viac zväčšuje.

Virtuálny DOM sa podobá na bežný DOM, ibaže si uchováva virtuálnu kópiu, aby ju mohol porovnať s aktualizovanou verziou. Tento proces sa nazýva *diffing* a v podstate sa vykonáva robí tak, že jedinou časťou DOM, ktorá sa aktualizuje, sú časti, ktoré sa zmenili namiesto aktualizácie celého stromu. [14]

Virtuálny DOM reaguje na zmeny premenných vstupujúcich do komponentu, ktoré voláme *props*. Tie môžu mať hodnotu od jednoduchých primitívnych typov až po zložitejšie ako sú napríklad funkcie. *Props* môžu byť zdieľané medzi viacerými komponentami a do rôznej hĺbky v strome našich komponentov.

Každá moderná aplikácia potrebuje vedieť reagovať na rôzne podnety. Napríklad, keď používateľ klikne na tlačidlo, otvorí sa modalové okno alebo sa zatvorí bočná lišta. Každou takouto zmenou môžeme povedať, že stav aplikácie (jej *state*) sa zmenil. Ten sa mení na základe dát, ktorými aplikácia disponuje. Keď sa spomenie pojem state

management, hovoríme o kombinácii dvoch vecí. Konkrétne o tom, ako aplikácia ukladá svoje dáta a ako ich mení. Najznámejším používaným state manažérom je Redux, ktorý ale postupne, kvôli svojim nevýhodám a nie najlepšej škálovateľnosti začína byť nahrádzaný funkcionalitou samotného Reactu prostredníctvom tzv. *hooks*, ktoré vedia vytvoriť dátové úložisko pre stav komponentu a zároveň aj jeho metódu *setState*, ktorá aktualizuje hodnotu stavu a DOM okamžite reaguje na zmeny.

Nasledujúca ukážka kódu na obrázku 7 využíva dva konkrétne *hooks* a to *React.useState* a *React.useEffect*. Metóda *useEffect* sa zavolá po zmene hodnôt, ktoré sú zapísané ako druhý argument v poli *dependencies*. V tomto prípade, keď sa zmení hodnota *newModule*, vykoná sa telo metódy, ktoré obsahuje ďalší *hook*, ktorému zmení predchádzajúci stav na aktuálny. Zmeny v strome sa vykreslia automaticky.

```
const [layouts, setLayouts] = React.useState(getInitLayouts());

React.useEffect( effect: () => {
  setLayouts( value: (prevState) => {
    const currentLayout = prevState[breakpoint]
    currentLayout.push(newModule.position)

    return {
      ...prevState,
      [breakpoint]: currentLayout
    }
  })
}, deps: [newModule]);
```

Obrázok 7: Príklad použitia dvoch *React Hooks*

Výhodou Reactu je jeho škálovateľnosť a možnosť opakovaného použitia komponentov, ktoré pri správnom návrhu stačí naprogramovať len raz a ich správanie meniť prostredníctvom zmien v *props*, ktoré do nich vstupujú, alebo priamo menením ich stavu. To mu umožňuje výborne škálovanie aj pri väčších projektoch a urýchlenie procesu vývoja. Vďaka svojej efektivite vykresľovania spríjemňuje skúsenosť s používaním pre svojich používateľov.

Ďalšou výhodou, ktorú nemožno nespomenúť je možnosť použiť React, konkrétne React Native pri tvorbe mobilných aplikácií. Jeden tím programátorov je tak veľmi jednoduché presúvať medzi rôznymi projektami bez nutnosti učenia sa nových postupov a programovacích jazykov.

4.1.2 React-grid-layout

Jedná sa o jedinú dostupnú knižnicu, ktorú sa nám podarilo nájsť, že spĺňa nasledujúce tri vlastnosti súčasne.

- ukladá komponenty do mriežky
- komponenty presúva metódou *drag and drop*
- dokáže meniť veľkosť komponentov

Celá knižnica je napísaná v Reacte a vychádza z dvoch pomocných knižníc, ktorými sú React-resizable a React-draggable. Rozloženie je responzívne a je zabezpečené zlomovými bodmi, ktoré sa dajú navoliť podľa šírky okna buď automaticky alebo prispôsobiť podľa potrieb používateľa. [15]

Knižnica zároveň disponuje možnosťou zarovnať elementy v mriežke horizontálne alebo vertikálne. Čo je však trochu prekvapujúce, tak pri snahe zarovnať elementy horizontálne a vertikálne zároveň, síce funguje podľa očakávaní, ale len do chvíle, kým sa nezmení veľkosť okna prehliadača. Vtedy rozmiestni elementy do mriežky prakticky náhodne. Predpokladáme, že vývojári knižnice to v budúcnosti napravia, ale vzhľadom na to, že táto funkcia bola pre nás dôležitá, jej dopracovanie zostalo našou úlohou.

4.1.3 Editor.js

Výber knižnice Editor.js sme zvolili podľa jej troch kľúčových vlastností. Ide o takzvaný *block-styled* editor, čiže namiesto klasického *wysiwyg* editoru, ktorý je tvorený jedným elementom umožňujúci úpravy svojho obsahu pomocou značkovacieho jazyka, sa Editor.js skladá z viacerých samostatných blokov, napr. odsekov, nadpisov, obrázkov, ale aj zložitejších ako Facebook *feeds*, *call-to-action* (CTA) tlačidiel a podobne. Každý z nich je nezávislý obsahovo upravovateľný prvok alebo štruktúra. Existuje niekoľko predpripravených blokov a vývojári knižnice pripravili aj jednoduché API na jej rozšírenie vytvorením vlastných blokov.

Rozdiel medzi týmito dvoma typmi editorov je vidieť aj vo výstupných dátach. *Wysiwyg* editor produkuje raw HTML s údajmi o obsahu a aj jeho vzhľadu. Naopak, Editor.js podáva ako výstupný JSON objekt s údajmi o každom bloku samostatne. Takýto

výstup sa dá ľahko použiť na vykreslenie pomocou HTML pre webových klientov, ale aj na vykreslenie pre natívne mobilné aplikácie. Údaje z objektu sa dajú dodatočne validovať a spracovať na serveri. Využitie by sa dokonca našlo aj na generovanie zvukovej verzie obsahu. [16]

4.2 Backend

Technológie na *backend-e* sú takisto zmenené oproti predošlému riešeniu. Podobne ako celý *frontend*, tak aj *backend* je tvorený z technológií písaných v JavaScripte, takže tu nedochádza k žiadnym zásadným syntaktickým zmenám. V tejto podkapitole sa pokúsime porovnať dôvod výberu NodeJS pred PHP a hlavné príčiny, kvôli ktorým sme sa rozhodli vymeniť relačnú databázu za nerelačnú.

4.2.1 Node.js

Node.js je webový server s architektúrou založenou na riadení udalostí. Úlohou webového servera môže byť napríklad otvorenie súboru na serveri a vrátenie obsahu klientovi. PHP spracováva žiadosť o súbor nasledujúco:

1. Odošle úlohu do súborového systému počítača
2. Po otvorení a prečítaní súborového systému server vráti obsah klientovi
3. Server je pripravený prijať ďalšiu požiadavku

Na druhej strane Node.js spracováva rovnakú žiadosť veľmi podobne, až na to, že hneď po druhom kroku je pripravený prijať ďalšiu požiadavku. Node.js tak eliminuje čakanie a jednoducho pokračuje ďalšou požiadavkou. [17]

Node pracuje s dvoma typmi vlákien. Jedným *Event loop* a viacerými *Worker-mi*. *Event loop* má na starosti počiatočnú inicializačnú fázu spojenú s importovaním modulov a registrovaním *callback* funkcií. Po skončení inicializačnej fázy vykonáva *callback* funkcie a je zodpovedný za riadenie neblokujúcich vstupno-výstupných operácií asynchrónnym spôsobom. *Worker* zas vykonáva úlohy, ktoré sú náročnejšie na procesor ako aj I/O, pre ktoré operačný systém neposkytuje neblokujúcu verziu. Oba typy vlákien pracujú naraz najviac na jednej požiadavke.

Node.js vyniká pri budovaní *real time* webových aplikácií, pretože dokáže požiadavky spracovať veľmi rýchlo a navyše s pomocou *web socketu* poskytuje obojstranné spojenie s klientom, to znamená, že aj server dokáže iniciovať komunikáciu. Naopak má problém so zložitými požiadavkami, ktoré silno zaťažujú CPU. Východzím správaním pri spracovaní takejto požiadavky je zablokovaný *Event loop,* ktorý prestáva obsluhovať ďalšie požiadavky. Čiastočným riešením je *partioning,* ktorý pravidelne dáva slovo čakajúcim udalostiam. Pri ešte väčšej záťaži je dobré použiť paradigmu *offloading,* ktorá presúva prácu z *Event loop* na *Workers,* ktoré majú ale takisto svoje kapacity. Preto je dôležité pri implementácii na to myslieť a snažiť sa používať radšej častejšie, ale o to menej náročné požiadavky.

NPM

Väčšina obslužného programu Node pochádza z jeho veľkej knižnice balíčkov. Po inštalácií Node.js je v príkazovom riadku prístupný príkaz *npm,* ktorý inštaluje tieto balíčky a podľa toho nastavuje súbor *package.json* v projekte. Node package management je správca s najväčším registrom balíčkov. Na svojej webovej stránke má viac ako milión dostupných balíčkov.

Nakoľko väčšina balíčkov pochádza z tretích strán, stojí za zváženie ich použitie. Treba si dať pozor a zamyslieť sa, či balíček nie je zbytočne komplikovaný a či nepoužíva príliš veľa ďalších závislostí na iné knižnice. [18]

4.2.2 MongoDB

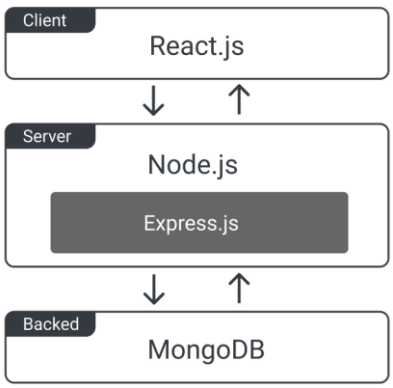
MongoDB je *open-source* projekt pre správu nerelačnej databázy (NoSQL), ktorý na spracovanie a ukladanie rôznych typov údajov používa namiesto tabuliek a riadkov dokumenty vo formáte binárny JSON. Umožňuje ľahké ukladanie a dopytovanie na viacrozmerné dátové typy, čím vývojárom zjednodušuje správu databázy a vytvára vysoko škálovateľné prostredie pre aplikácie a služby naprieč platformami. V prípade, že by sa jednalo o väčší systém, tak dokáže vyrovnávať zaťaženie distribuovaním veľkých množín údajov na viac virtuálnych strojov naraz, pričom si stále zachováva prijateľnú priepustnosť na čítanie a zápis. Týmto horizontálnym škálovaním (tzv. *sharding*), pomáha znížiť náklady spojené s vertikálnym škálovaním hardvéru. [19]

V protiklade mu stojí MySQL, ktorý používa na prístup k uloženým údajom v tabuľkách štruktúrovaný dopytovací jazyk. Schémy na vytváranie databázových štruktúr, musia byť dopredu definované a obvykle zostávajú nemenné. Na to aby mohli byť dáta uložené do tabuliek, musia sa presne zhodovať s jej schémou. Z dôvodu svojej rigidnej povahy je MySQL vhodnejšia ako MongoDB v prípadoch, keď je nevyhnutná integrita a izolácia údajov. Bežný príklad takéhoto prípadu môže byť bankový systém, ktorý požaduje silnú referenčnú integritu, spojenú s transakciami a uchovávaním jednotných dátových typov. V prípade, že by však bolo potrebné uložiť nový typ alebo formát do už existujúcej databázy, bola by potrebná migrácia schém, ktorej náročnosť stúpa s veľkosťou použitej databázy. [19]

Vzhľadom na dynamický návrh, ktorý poskytuje MongoDB je z nášho pohľadu vhodnejší pre tvorbu CMS, pretože ľahko dovolí pridávať nové funkcie a atribúty na rozšírenie pôvodného návrhu.

4.3 MERN

Spomenuté technológie spojením dokopy vytvárajú takzvaný MERN *stack*, čím vzniká trojvrstvová architektúra, kde sa na každej vrstve používa Javascript a JSON. Preto je pre programátora vývoj za pomoci MERN-u oveľa jednoduchší, lebo mu stačí ovládať menšie množstvo technológií na vysokej úrovni. Tiež je to výhoda pre začínajúcich programátorov. S Node.js spolupracuje *framework* Express.js, ktorý očakáva od klienta požiadavky na základe URL adresy a posiela klientovi naspäť príslušnú odpoveď. Obrázok 8 znázorňuje smer komunikácie medzi jednotlivými vrstvami.



Obrázok 8: Smer komunikácie trojvrstvovej architektúry MERN

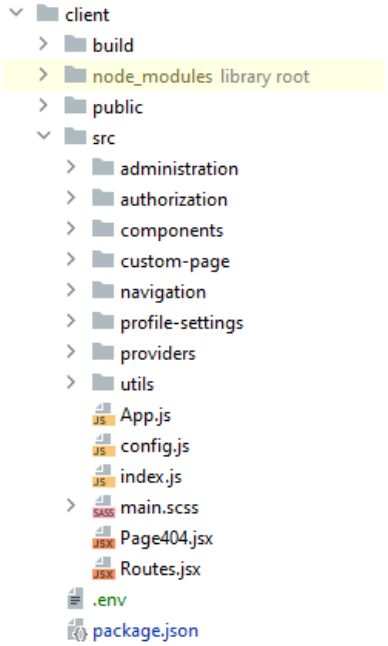
5 Implementácia

Skôr ako sme sa pustili do implementácie na základe nášho návrhu, vytvorili sme si na gite repozitár. Git sleduje zmeny vykonané v súboroch, takže sme mali lepší prehľad o tom, čo sa urobilo a v prípade potreby sme sa mohli vrátiť ku konkrétnej verzii naspäť.

5.1 Štruktúra súborov

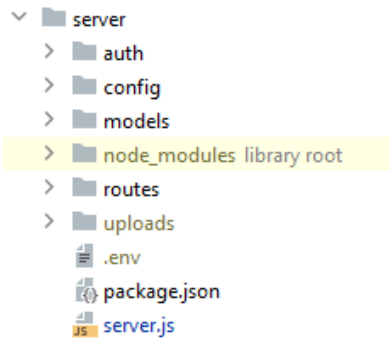
Samotný projekt je zložený z dvoch menších aplikácií. Implementácia webu v Reacte sa nachádza v priečinku *client* a backend využívajúci Node.js je v priečinku *server*. Všetky použité balíčky sú nainštalované v priečinkoch *node-modules*. V obidvoch našich aplikáciách sa nachádza aj *.env* súbor, ktorý obsahuje konštanty potrebné pre spustenie aplikácie v rôznych prostrediach.

Na obrázku 9 vidno, že c*lient* je rozdelený na ďalšie priečinky. V *public* sú statické súbory ako napríklad obrázky. Samotný kód sa nachádza v priečinku *src* a je ďalej členený podľa postupne pribúdajúcich potrieb. V zásade sme sa ale snažili, aby každá stránka mala svoj vlastný priečinok v ktorom sú súbory s jej vlastnými komponentmi. Komponenty ktoré sú použité naprieč viacerými stránkami sú v priečinku *components*. V priečinkoch *providers* a *utils* sú pomocné .js súbory, ktoré sa rovnako využili vo viacerých súboroch.



Obrázok 9: Štruktúra súborov pre frontend

Server má okrem hlavného súboru *server.js* takisto viacero priečinkov. V *config* sa nachádzajú nastavenia pre spojenie s databázou. *Auth* obsahuje funkcie na autorizáciu používateľa. V *models* sa nachádzajú definície schém pre jednotlivé entity, ktoré sú v databáze. R*outes* obsahuje GET a POST metódy, ktoré obsluhujú požiadavky z klienta na základe URL a prenesených dát. Do posledného priečinku *uploads* sa ukladajú všetky nahraté súbory, ktoré používatelia pri editovaní stránky pridali do ich obsahu. Štruktúru súborov pre backend vidno na nasledujúcom obrázku 10.



Obrázok 10: Štruktúra súborov pre backend

5.2 Custom page

Custom page sme implementovali tak, že sa dá vyskladať z rôznych typov modulov, ktoré svoju pozíciu ukladajú do mriežky. Mriežka je implementovaná pomocou knižnice React-grid-layout. V našom prípade sme chceli použiť jej responzívnu variantu. Medzi povinné *props*, ktoré treba definovať pre responzívny komponent *ResponsiveGridLayout* patrí počet stĺpcov (*cols*), zlomové body (*breakpoint*s) a šírka (*width*). Podľa nich vie knižnica určiť kedy má zmeniť rozloženie, ktoré sa zobrazí. Dokumentácia odporúča použiť na meranie šírky jej pripravený komponent *WidthProvider*, ktorý obsahuje funkcionalitu zabezpečujúcu správne zalamovanie obsahu pri rôznych šírkach.

V našom prípade sme potrebovali vedieť meniť šírku mriežky aj ručne, aby editor stránky pri testovaní toho ako vyzerajú rôzne rozloženia nemusel meniť šírku len ťahaním hrany okna. Vytvorili sme preto panel nástrojov s tlačidlami, ktoré kliknutím zmenia šírku a tým zmenia prislúchajúce rozloženie. Konkrétne sme vytvorili tri tlačidlá, ktoré simulujú zariadenie telefónu, tabletu alebo desktop. Na obrázku 11 je zdrojový kód komponentu *ViewToolbar*.

```
const ViewToolbar = () => {
  // render one of the three buttons
  const ViewButton = ({ breakpoint, active, disabled, children }) => (
    <SmallButton
      variant="outline-dark"
      active={active}     // to mark pressed button
      disabled={disabled} // disable button with smaller width that the current
      onClick={() => setWidth( value: breakpoints[breakpoint] + 1)}
    >
      {children}          {/* device name */}
    </SmallButton>
  );

  return (
    <div className="d-flex justify-content-center pb-4">
      {/* render group of three buttons for mobile, tablet and desktop  */}
      <ButtonGroup...>
    </div>
  );
};
```

Obrázok 11: komponent *ViewToolbar* vykresľujúci panel nástrojov s tlačidlami

Problém bol však v tom, že *ResponsiveGridLayout* ignoroval *prop width*, aj napriek tomu, že by sa podľa dokumentácie mal dať vedieť zadať. Zistili sme, že tým, že sme použili *WidthProvider*, tak *ResponsiveGridLayout* svoju šírku nevie zmeniť zvonku. Preto sme sa rozhodli, že si šírku budeme riadiť sami. Použili sme knižnicu React-sizeme, ktorej komponent automaticky deteguje zmenu šírky svojho dieťaťa a vracia jej hodnotu pri každej zmene. Použitie komponentu SizeMe je na obrázku 12. Podľa neho sme vždy vedeli aká je aktuálna šírka *ResponsiveGridLayout* a vkladali sme ju ako *prop*. Získali sme tým také isté správanie ako sme mali predtým, ale tento krát sa dala meniť šírka aj kliknutím na tlačidlá.

```
  return (
    <div>
      {simulateBreakpoints && <ViewToolbar />}
      <SizeMe>
        {({ size : {readonly height: number | null, readonly width: number | null} }) => {
          handleSizeChange(size);
          return (
            <ResponsiveGridLayout
              width={width}
              breakpoints={breakpoints}
              compactType={compactType}
              measureBeforeMount={!animated}
              useCSSTransforms={animated}
              isResizable={isResizable}
              isDraggable={isDraggable}
              onDragStop={onDragStop}
              onResizeStop={onResizeStop}
              rowHeight={ROW_HEIGHT}
              {...rest}
            >
              {children}
            </ResponsiveGridLayout>
          );
        }}
      </SizeMe>
    </div>
  );
```

Obrázok 12: Komponent vykresľujúci mriežku pre *Custom page*

Problémom ešte zostalo, že po kliknutí na tlačidlo v panely sa šírka síce zmenila, ale hneď sa vrátila na skutočnú šírku *ResponsiveGridLayout* podľa nameranej hodnoty z komponentu *SizeMe*. Zabránili sme tomu funkciou, ktorá je na obrázku 13. V nej ukladáme predchádzajúcu vypočítanú šírku okna. V prípade, že sa nezmenila oproti predošlej šírke, tak nenastavujeme novú, tým pádom sa šírka nenastaví na novú a tak ostane šírka podľa stlačeného tlačidla.

```
const handleSizeChange = (size) => {
  if (prevSize.width === size.width) return;

  setWidth(size.width);
  setPrevSize(size);
};
```

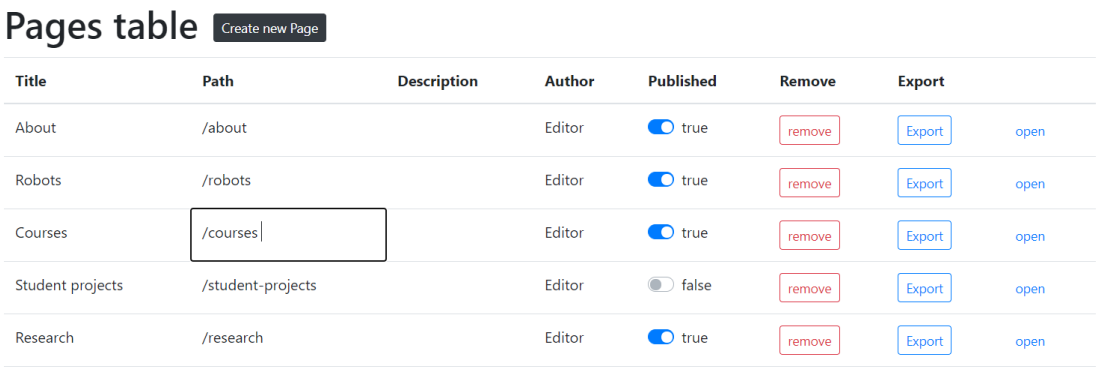
Obrázok 13: Funkcia na nastavenie novej šírky z komponentu *SizeMe*

6 Výsledky práce

6.1 Administrácia

Prihlasovanie, registráciu a zmenu profilových údajov, sme prakticky len preprogramovali do našich použitých technológii. Jedinou zmenou, bolo sprístupnenie registrácie nového používateľa len administrátorovi.

Administrácia stránok a používateľov ostala reprezentovaná v tabuľkách. S predošlým riešením sa trochu líšia v definovaných stĺpcoch. Na obrázku 14 je vidno, že pre tabuľku so stránkami sa hodnoty v stĺpcoch *Title, Path a Description* dajú meniť priamo v bunke, bez toho aby bolo potrebné otvárať modálne okno s formulárom. Keď používateľ takýmto spôsobom zmení hodnotu, tak sa v pravom hornom rohu ukáže notifikácia o úspešnosti vykonania. Dôvodom je hlavne to, že ak by zmenená hodnota nespĺňala validáciu, tak aby sa o tom používateľ hneď dozvedel. V takom prípade sa nesprávny údaj neuloží do databázy a hodnota v bunke sa automaticky vráti do pôvodného stavu.



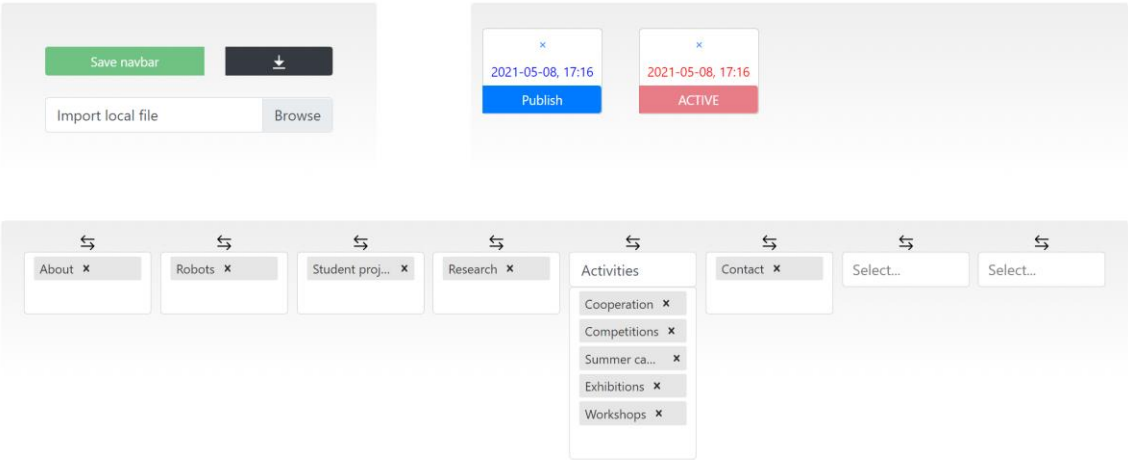
Obrázok 14: : Modifikovanie tabuľky so zoznamom všetkých vytvorených stránok

6.2 Navigácia

Oproti predchádzajúcemu riešeniu sa nám výrazne podarilo vylepšiť nastavenia navigácie. Predošlé riešenie dovoľovalo pridávať odkazy na stránky do navigácie bez možnosti určenia ich poradia. Okrem toho bolo jedno miesto v navigačnom paneli napevno rezervované pre *dropdown*.

Od pôvodného dizajnového návrhu sme sa čiastočne odklonili, pretože nastavovanie priamo v navigačnom panely spôsobovalo komplikácie, ktoré sme rozumným spôsobom nevedeli vyriešiť. Navyše sme k pôvodne zamýšľanej funkcionalite pridali aj import a export a aj jednoduchý systém na spravovanie verzií. Kvôli tomu sme sa rozhodli, že pôvodne zamýšľané nastavovanie v navigačnom panely presunieme na samostatnú stránku.

V konečnom výsledku, ktorý je zobrazený na obrázku 15, sa odkazy na stránky dajú horizontálne presúvať, čím sa mení ich poradie, v ktorom budú zobrazené. Takisto je možné na ľubovoľnej pozícii vytvoriť *dropdown* pridaním viacero odkazov na jedno miesto a rovnako sa s ním dá presúvať do strán. Po kliknutí na tlačidlo *„Save navbar“* sa do databázy nespraví update, ale uloží sa nový záznam, preto je možné vrátiť sa k predchádzajúcim verziám. V prípade, že by stránka na ktorú je vytvorený odkaz už neexistovala, systém ju odignoruje aby nespôsobovala problémy. V prípade, že sa v navigácii odkazuje na nepublikovanú stránku, tak sa v robrazení pre návštevníka takisto ignoruje.

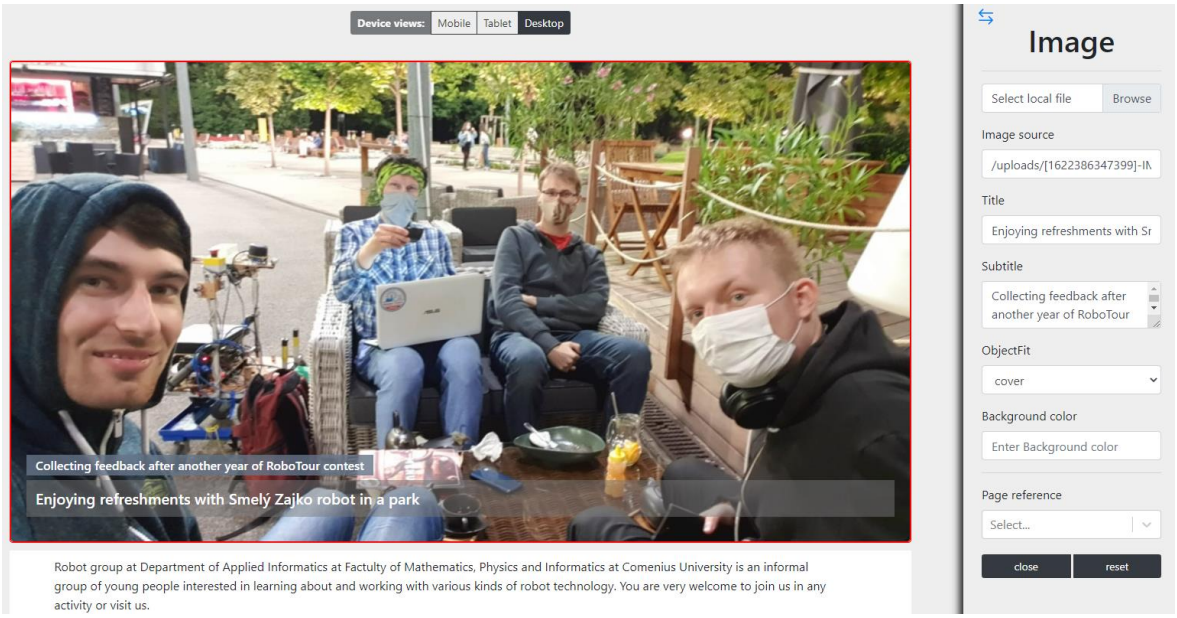


Obrázok 15: Výsledok stránky, na ktorej sa dá spravovať navigácia

6.3 Vytváranie stránok

Hlavnou funkcionalitou aplikácie je vytváranie stránok. To sa nám podarilo naprogramovať podľa predpokladaného návrhu. *Custom page* obsahuje dva panely nástrojov. Ako vidno na obrázku 16, jeden je naľavo s akciami pre náhľadový režim, nastavenia, možnosťou vytvorenia novej stránky, exportom a tlačidlom na uloženie zmien.

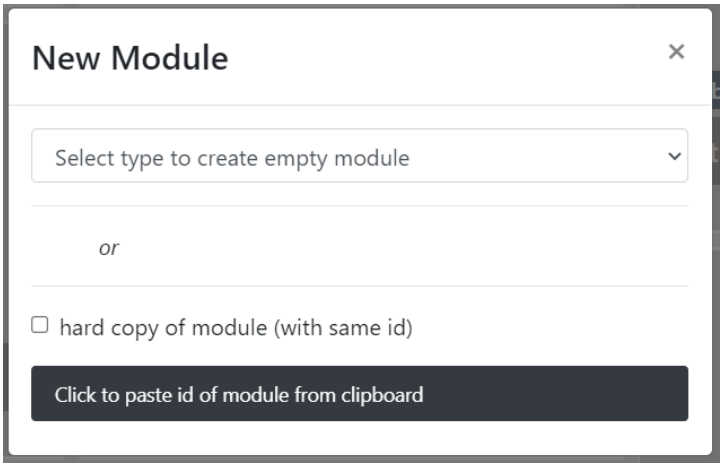
Horný panel nástrojov ako sme už opisovali je na rôzne pohľady a úplne napravo sa nachádza bočný panel s formulárom pre úpravu práve označeného (aktívneho) modulu.



Obrázok 16: Pohľad na ukážku vytvárania stránky

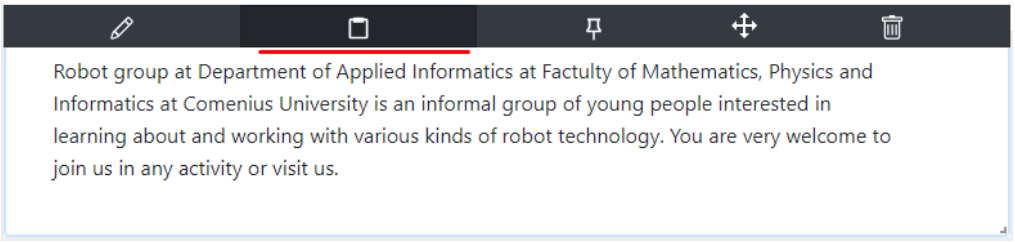
6.4 Moduly

Pri vytváraní modulu sa otvorí rovnaký modál ako je na obrázku 17. Používateľovi dávame na výber zo šiestich typov modulov*: Image, Html, Video, Gallery, Editor-js* a *Alert*. Hneď ako vo formulári vyberie niektorý typ, modálne okno sa zavrie a na poslednú pozíciu v mriežke pridá modul daného typu s prázdnym obsahom. Používateľ okrem výberu z možností typov môže vytvoriť nový modul aj vytvorením kópie existujúceho modulu kliknutím na spodné tlačidlo v obrázku.



Obrázok 17: Modálne okno na vytvorenie modulu

Operácia, ktorá sa vykoná na pozadí je, že vloží skopírované id modulu z používateľovej schránky. Používateľ si tento údaj do schránky vloží kliknutím na ikonu, ktorá je označená na obrázku 18. Zároveň si môže zvoliť, či sa má nový modul vytvoriť s novým alebo rovnakým id. Ak by sa modul vytvoril s rovnakým id, znamenalo by to, že zmeny vykonané na module v stránke A by sa objavili aj na stránke B. Toto môže byť užitočné, ak by chcel mať používateľ na dvoch stránkach ten istý komponent.



Obrázok 18: Panel nástrojov, ktorý sa zobrazí nadídením myšou ponad modul

Existuje aj rýchlejšia cesta ako vytvoriť kópiu bez rovnakého id. Dvojitým kliknutím na tú istú ikonu z obrázka 18 sa do mriežky pod pozíciu modulu pridá nový modul, ktorý bude mať identický obsah. Takýto scenár má využitie, keď používateľ potrebuje vytvoriť viacero podobných modulov.

6.5 Single page app

Celkovým výsledkom práce je *single page* aplikácia, ktorá si potrebné dáta zo servera vypýta pri prvom načítaní stránky a zvyšok riadi Javascript na strane klienta. Pri prekliknutí na ďalšiu stránku, tak používateľ nemusí čakať na html zo servera a takmer okamžite dostane zobrazený jej obsah.

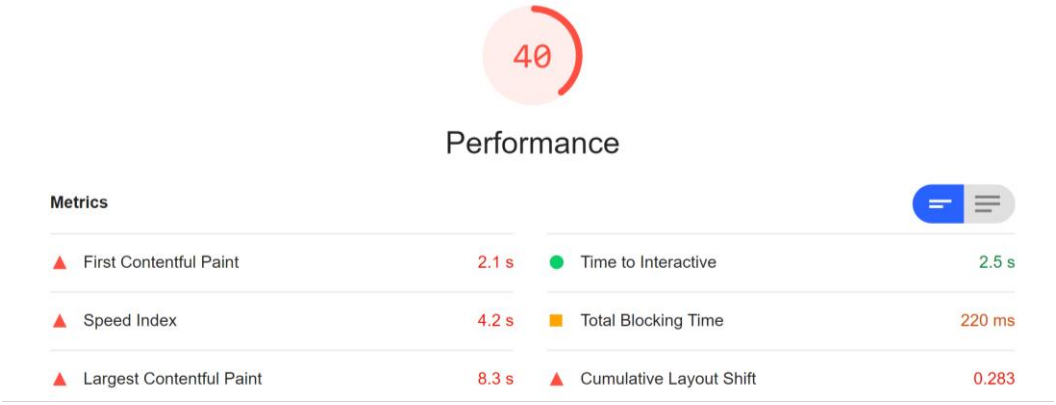
Použitím knižnice react-router-dom sme dosiahli, že používateľa pri kliknutí na odkaz v rámci aplikácie v skutočnosti nepresmeruje na ďalšiu stránku, ale len prekreslí obsah tej aktuálnej. Preto je *single page* aplikácia naozaj len jedna jediná stránka aj napriek tomu, že sa v prehliadači mení URL a vytvára história. React-router-dom poskytuje komponent *Link* namiesto klasického *anchor* tagu z HTML. URL je spracovaná v oboch prípadoch podobne až na to, že react-router-dom nespôsobí presmerovanie na ďalšiu stránku,

ale prekreslí obsah na tej istej stránke podľa komponentu, ktorý je definovaný k príslušnej adrese podľa regex výrazu.

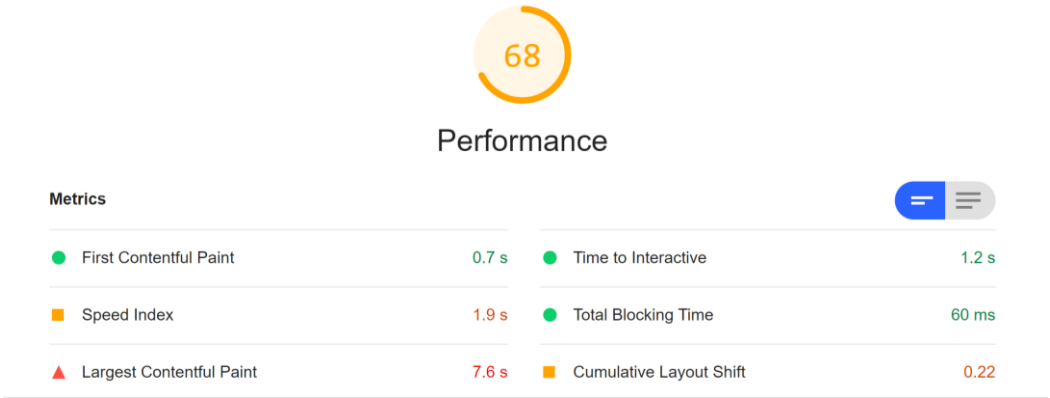
6.6 Test výkonnosti stránky

Na zistenie výkonu našej webovej aplikácie sme vyskúšali *Lighthouse* test, ktorý sa nachádza v prehliadači Chrome. Test spočíva v tom, že prehliadač začne načítať stránku a spustí nahrávanie videa, z ktorého si spätne šiestimi metrikami vyhodnotí za ako dlho sa načítal obsah a možnosť interakcie so stránkou. Pre nezávislé výsledky z merania obmedzí rýchlosť internetu na 10,240*Kbps* a odporúča použiť inkognito režim bez využitia cache pamäti.

Na obrázku 19 je výsledok testovania pred tým, ako sme použili príkaz *npm run build*, ktorý skomprimuje zdrojový kód pre klienta na minimum. Obrázok 20 zachytáva výsledok po komprimácii súborov vykonaným spomenutým príkazom.

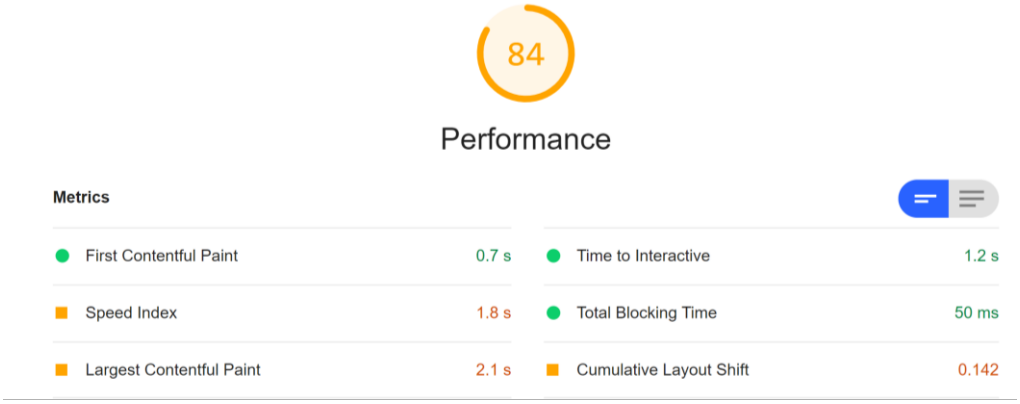


Obrázok 19: Výkon bez skomprimovaného zdrojového kódu



Obrázok 20: Výkon po komprimácii zdrojového kódu

Výsledky z testovania sa zlepšili o 14% po tom čo sme skomprimovaním zmenšili veľkosť obrázkov. Okrem toho sme použili *React.Suspend*, ktorý namiesto komponentu čakajúceho na stiahnutý obsah, vykreslí náhradný komponent až kým sa nedokončí sťahovanie. *React.Suspend* však nijak neovplyvnil výsledok, pretože metrika merajúca čas prvého vykresleného obsahu sa nezmenila. Výsledky z posledného testovania zachytáva nasledujúci obrázok 21.

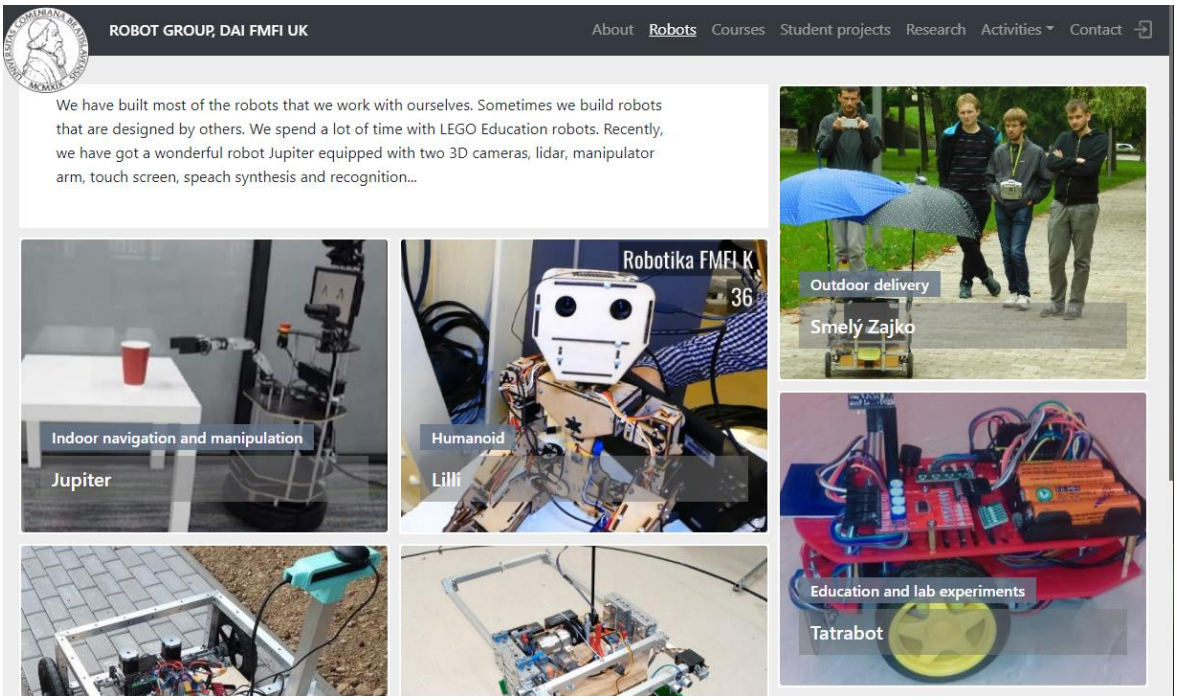


Obrázok 21: Výsledný výkon aplikácie

6.7 Prezentácia robotickej skupiny

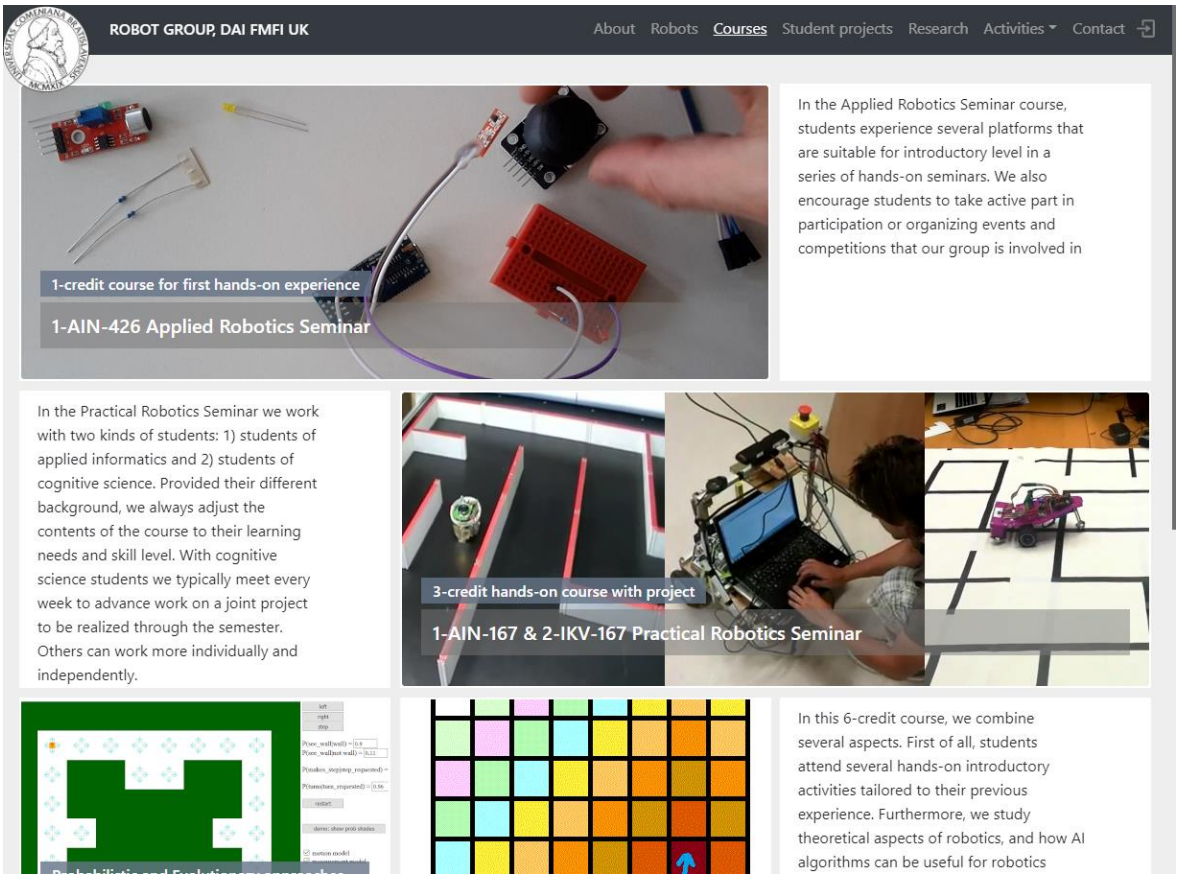
Do výslednej aplikácie sme vytvorili obsah na prezentáciu robotickej skupiny na našej katedre. Počas vytvárania sme odhalili ešte zopár menších chýb, ktoré sme opravili a aktualizovali aplikáciu na novú verziu. Stránka je publikovaná na webovej adrese https://kempelen.dai.fmph.uniba.sk/robots/.

Cesta */robots* slúži na prezentáciu jednotlivých typov robotov. Vďaka modularite navrhnutého systému bude jednoduché pridávať nových robotov a upravovať umiestnenie ich zobrazenia podľa toho, ako to bude v danom momente relevantné.



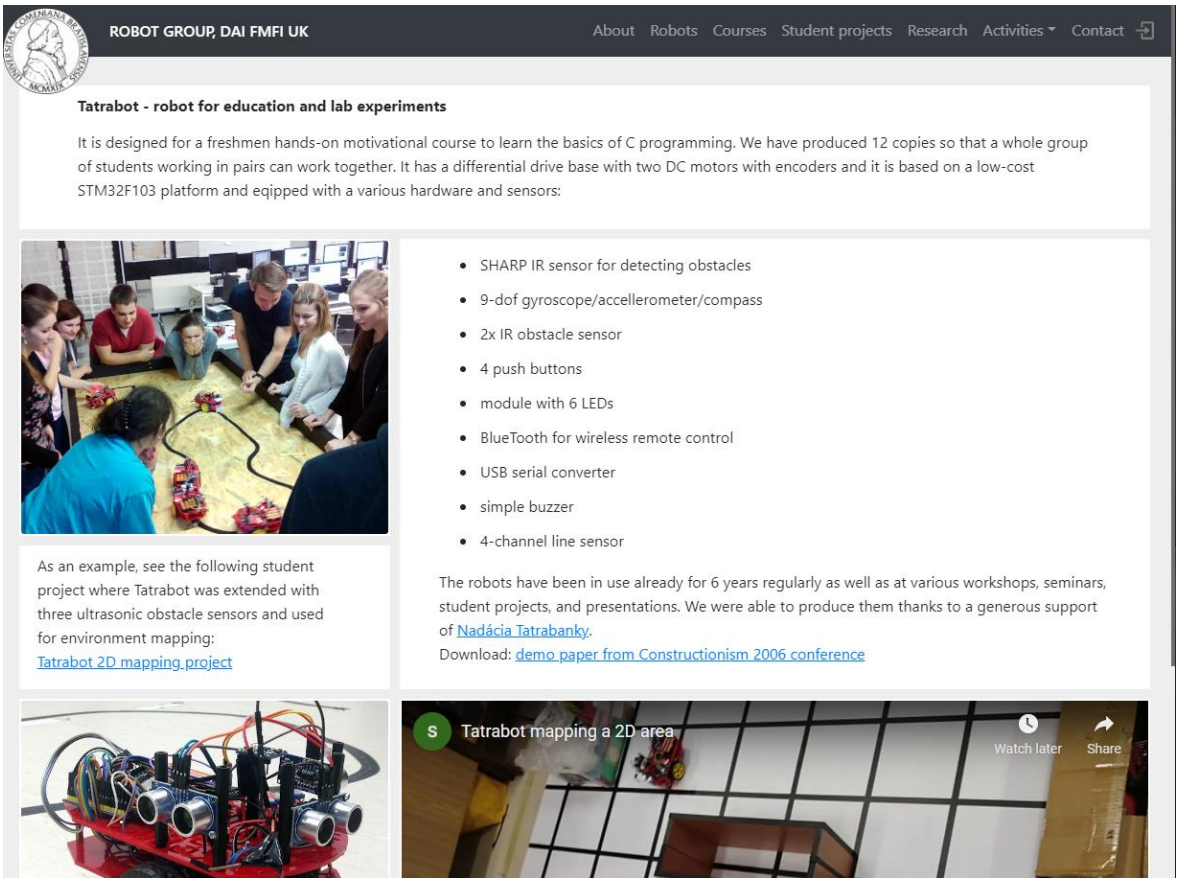
Obrázok 22: Stránka zobrazujúca rôznych robotov, ktorými disponuje naša katedra

Iným príkladom rôzneho rozloženia modulov je stránka na ceste /courses. Poskytuje prehľad o predmetoch nasadených do jednotlivých študijných programov našej katedry, ktoré sa týkajú robotiky. Ich súčasťou sú rôzne praktické úlohy či cvičenia. Usilovali sme sa, aby na vytvorených stránkach bola prehľadne a atraktívne zobrazená práca na katedre. Zároveň tieto stránky poskytujú užitočné testovacie prostredie na doladenie drobných chýba prípadné objavenie novej funkcionality, ktorá by mohla v budúcnosti nájsť uplatnenie na ďalších stránkach.



Obrázok 23: Stránka kurzov robotiky

Peknou ukážkou prepojenia jednotlivých celkov je možné vidieť na obrázkoch 22 a 24, kde budeme po kliknutí na modul s Tatrabotom na obrázku 22 presmerovaní na stránku zobrazenú na obrázku 24. Tam môžeme vidieť použitie knižnice Editor.js, ktorá nám umožnila vytvárať bohatšie texty aj vďaka možnosti používať nadpisy, podnadpisy a rôzne číslované alebo bodované zoznamy.



Obrázok 24: Stránka venovaná robotovi Tatrabot

Záver

V bakalárskej práci sa nám podarilo splniť stanovený cieľ práce. Získali sme množstvo poznatkov o fungovaní CMS a jeho vlastnej implementácii pomocou MERN-u. Celkovým výsledkom práce je aplikácia bežiaca na školskom serveri. Na jej adrese sa nachádza obsah robotickej skupiny. URL adresa je publikovaná aj na stránke katedry. Okrem toho je výsledkom open-source aplikácia, ktorú je možné použiť na vytvorenie prezentácie ľubovoľnej výskumnej skupiny. Zdrojový kód sa nachádza na https://github.com/Robotics-DAI-FMFI-UK/researchgroupweb.

Najdôležitejšou súčasťou práce bola samotná implementácia riešenia. Tá prebiehala vo viacerých krokoch. Začala sa analýzou existujúceho riešenia, ktorým disponuje robotická skupina a ostatných známych CMS. Potom sa úsilie presunulo ku grafickému a architektonickému návrhu, ktoré zabezpečili to, že veľká časť funkcionality bola premyslená ešte pred samotným začatím programovania.

Hlavným cieľom tejto práce bolo vytvoriť webovú aplikáciu, ktorá by umožnila tvorbu vlastného obsahu. Dôležitým parametrom na splnenie zadania bolo aj to, aby mal výsledný produkt moderný dizajn a bol responzívny.

Výsledná webová aplikácia má ešte stále priestor na vylepšovanie a v budúcnosti by sme sa radi venovali možnostiam ako pridať k jednotlivým článkom diskusiu, pridať prepracovanejší manažment rolí a rozšíriť paletu modulov.

Zoznam použitej literatúry

[1] A. Osmani, Learning JavaScript Design Patterns, Sebastopol: O'Reilly Media, Inc, USA, 2012.

[2] Codeacademy, „MVC: Model, View, Controller,“ 2021. [Online]. Available: https://www.codecademy.com/articles/mvc. [Cit. 3 Február 2021].

[3] Developers, Chrome, „MVC Architecture,“ 21 Máj 2014. [Online]. Available: https://developer.chrome.com/docs/apps/app_frameworks/. [Cit. 3 Február 2021].

[4] SWED Team, 2015. [Online]. Available: https://github.com/SWED-team/TIS. [Cit. 27 Január 2021].

[5] W3Techs, „World Wide Web Technology Surveys,“ 2021. [Online]. Available: https://w3techs.com/. [Cit. 4 Apríl 2021].

[6] Adobe, „Content management system (CMS),“ 2021. [Online]. Available: https://business.adobe.com/glossary/content-management-system.html. [Cit. 4 Apríl 2021].

[7] V. J. Kadam a . e. a. , „How to Choose a Website Content Management System,“ Conference: National Conference On Emerging Trends In Academic LibraryAt: SSVP's Dr. P.R. Ghogrey Science College, 18.-19. Január 2013. [Online]. Available: https://www.researchgate.net/publication/274510334_How_to_Choose_a_Website_Content_Management_System. [Cit. 5 Apríl 2021].

[8] J. Garcia, „10 Stats That Show the Size of Wix’s Market Share,“ 26 Február 2020. [Online]. Available: https://www.websitetooltester.com/en/blog/wix-market-share/. [Cit. 4 Apríl 2021].

[9] E. Gardner a A. Tomasevich, „2020: The Year in Vue,“ 22 Január 2021. [Online]. Available: https://techblog.wikimedia.org/2021/01/22/2020-the-year-in-vue/. [Cit. 28 Apríl 2021].

[10] Onejohi, „Offline Web Apps: Using Local Storage and Service Workers.,“ 13 Júl 2018. [Online]. Available: https://medium.com/@onejohi/offline-web-apps-using-local-storage-and-service-workers-5d40467117b9. [Cit. 18 Máj 2021].

[11] J. Tucker, „Next.js (SSR) vs. Create React App (CSR),“ 11 December 2018. [Online]. Available: https://codeburst.io/next-js-ssr-vs-create-react-app-csr-7452f71599f6. [Cit. 11 Máj 2021].

[12] K. Shah, „Client-side Vs. Server-side Rendering: What to choose when?,“ 29 November 2019. [Online]. Available: https://www.solutelabs.com/blog/client-side-vs-server-side-rendering-what-to-choose-when. [Cit. 11 Máj 2021].

[13] C. Cousins, „What Is Figma?,“ 20 November 2019. [Online]. Available: https://designshack.net/articles/software/what-is-figma-intro/. [Cit. 3 Máj 2021].

[14] A. Richards, „Why You Should Learn React Instead of jQuery,“ 1 Apríl 2020. [Online]. Available: https://javascript.plainenglish.io/why-you-should-use-react-instead-of-jquery-d68b5b129bbb. [Cit. 4 Máj 2021].

[15] React grid layout, „React grid layout,“ 8 December 2016. [Online]. Available: https://github.com/react-grid-layout/react-grid-layout. [Cit. 12 Apríl 2021].

[16] Editor.js, „Editor.js,“ 5 Jún 2020. [Online]. Available: https://editorjs.io/. [Cit. 17 Máj 2021].

[17] W3School, „Node.js Introduction,“ 2021. [Online]. Available: https://www.w3schools.com/nodejs/nodejs_intro.asp. [Cit. 17 Marec 2021].

[18] Node.js, „Don't Block the Event Loop (or the Worker Pool),“ [Online]. Available: https://nodejs.org/en/docs/guides/dont-block-the-event-loop/. [Cit. 22 Apríl 2021].

[19] IBM, „What is MongoDB?,“ 21 December 2020. [Online]. Available: https://www.ibm.com/cloud/learn/mongodb. [Cit. 22 Apríl 2021].