VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE PRO VYTVÁŘENÍ RELAČNÍCH DATABÁZOVÝCH SCHÉMAT

WEB APPLICATION FOR CREATING OF RELATIONAL DATABASE SCHEMAS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ROMAN ORSZÁGH**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148572

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Országh Roman**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Webová aplikace pro vytváření relačních databázových schémat**
Kategorie:	Databáze
Akademický rok:	2022/23

Zadání:

1. Seznamte se s principy tvorby webových aplikací, relačními databázemi a webovými sockety.
2. Analyzujte současné aplikace pro vytváření relačních databázových schémat, zhodnoťte jejich výhody a nevýhody.
3. Navrhněte webovou aplikaci pro vytváření relačních databázových schémat a export do migračních skriptů, která bude zahrnovat editor diagramů schémat databáze, možnost tvorby databáze pomocí pseudokódu a možnost vzdálené kooperace na vytváření schématu.
4. Po konzultaci s vedoucím implementujte navrženou aplikaci a ověřte její funkčnost na vhodném vzorku dat.
5. Zhodnoťte dosažené výsledky a další možnosti v pokračování tohoto projektu.

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.
- Lombardi, A.: WebSocket. O'Reilly Media, Inc, USA, 2015. ISBN 1449369278.

Při obhajobě semestrální části projektu 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:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	24.10.2022

Abstrakt

Cieľom tejto práce je vytvorenie webovej aplikácie pre tvorbu relačných databázových schém a vytváranie migračných kódov pre PHP aplikačný rámec Laravel. Databázové schémy môžu byť vytvorené za pomoci jednoduchých vstupných políčok alebo pomocou navrhnutého pseudokódu, ktoré sa potom automaticky generujú do jednotlivých súborov s využitím štruktúry Laravelu. Aplikácia je na frontende implementovaná s využitím knižnice React.js a PHP aplikačný rámec Laravel, ktorý slúži ako backend. Celá aplikácia je ovládateľná prostredníctvom jednoduchého a intuitívného užívateľského prostredia, ktorá obsahuje jednotlivé projekty reprezentujúce databázové schémy. Vytvorené databázové schémy zahrňujú všetky potrebné tabuľky a ich definované stĺpce. Aplikácia taktiež umožňuje vzdialenú kooperáciu pomocou web soketov.

Abstract

The goal of this work is to make a web application for creating relation database schemas and migration files for PHP framework Laravel. Relation databases can be created by using simple input fields or via pseudo-code, which are then generated into individual files into the Laravel like structure. React.js is used as a front-end of the web application, and PHP Framework Laravel as back-end. The whole application is manageable by simple and intuitive user interface, which contains individual projects which represents database schemas. Created database schemas include all required tables and their defined columns. Web application also offers remote cooperation via web sockets

Kľúčové slová

web, webová aplikácia, databázové schémy, migračné skripty, pseudokód, vzdialená kooperácia, React, Next.js, Laravel, MySQL, sockety

Keywords

web, web application, database schemes, migration scripts, pseudocode, remote cooperation, React, Next.js, Laravel, MySQL, sockets

Citácia

ORSZÁGH, Roman. *Webová aplikace pro vytváření relačních databázových schémat.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Webová aplikace pro vytváření relačních databázových schémat

Prehlásenie

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

.....
Roman Országh
3. mája 2023

Poďakovanie

Ďakujem svojmu vedúcemu práce Ing. Vladimírovi Bartíkovi, Ph.D. za pomoc a odbornú pomoc poskytnutnú pri vypracovávaní tejto práce. Taktiež by som chcel poďakovať všetkým, ktorí sa podielali na testovaní mojej aplikácie.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

V dnešnej dobe je dôležité robiť všetko, čo najrýchlejšie a najefektívnejšie. Existuje veľa nástrojov pre zvýšenie produktivity práce či už sú to rôzne aplikácie schopné riešiť viaceré vecí naraz, automatizácie alebo umelá inteligencia.

Pri vývoji webových aplikácií je vo väčšine prípadoch nutné pracovať aj s databázami, ktoré poskytujú úložisko údajov zobrazované na webovej stránke či už je to obsah stránky, informácie o užívateľovi, alebo články. Na vytváranie štruktúry databáz sa tvoria rôzne diagramy a modely. Pre ich lepšiu vizualizáciu, sa používajú rozličné aplikácie, ktoré sú schopné tieto databázové schémy generovať.

Webové technológie idú rýchlo dopredu a pribúda čoraz viac aplikačných rámcov, ktoré pomáhajú pri vývoji. Medzi najpopulárnejšie v programovacom jazyku PHP patrí aplikačný rámec Laravel. Táto bakalárska práca sa zaoberá práve webovou aplikáciou, ktorá rieši efektívnosť vytvárania databáz, migráciu dát a tvorbu modelov práve pre Laravel. Postupy pri návrhu backendu webovej aplikácie zvyčajne začínajú návrhom databázovej štruktúry, aby sa mohli ďalej jednoducho implementovať všetky potrebné funkcie. Pri Laraveli sa po návrhu databázy napíšu jednotlivé migračné súbory a k ním aj modely. Aplikácia v praktickej časti tejto bakalárskej práce tieto postupy zjednoduší tým, že už pri návrhu databázy dokáže potom vygenerovať potrebné súbory, ktoré jednoducho stačí len presunúť do Laravel aplikácie. Hlavným dôvodom tejto aplikácie je práve zjednodušenie a urýchlenie procesu vytvárania týchto súborov. Okrem spomínaných súborov, ktoré bude generovať, bude taktiež podporovať vzdialenú kooperáciu pomocou web soketov.

Kapitola 2

Webové aplikácie a ich vývoj

Táto kapitola sa zaoberá všeobecným popisom webových aplikácií a ich vývojom. Taktiež popisuje rozdelenie webových aplikácií a ich plusy a mínusy. Jednotlivé postupy pri ich vyvíjaní sú rozdelené do troch hlavných častí. Prvou časťou je klientska časť alebo frontend, ktorá sa zaoberá tým, čo užívateľ vidí, keď vstúpi na webovú stránku. Druhá časť je serverová časť alebo backend, ktorá spracováva požiadavky, ktoré užívateľ zadáva či už požiadaním o načítanie konkrétnej stránky, alebo rôznymi vstupmi z formulárov a pod. Tretia časť popisuje typ komunikácie medzi frontendom a backendom. Posledná časť tejto kapitoly spomína vývojové nástroje, ktoré boli využité pri implementovaní praktickej časti bakalárskej práce.

2.1 Rozdelenie webových aplikácií

Webové aplikácie sa rozdeľujú na statické a dynamické. Statické aplikácie sú tie, ktoré neobsahujú backend, resp. nemenia obsah dynamicky. Zdrojový kód je odoslaný v odpovedi od servera rovno bez akýchkoľvek zmien. Takéto aplikácie nemajú žiadnu interakciu s databázou. Naopak dynamické webové aplikácie využívajú backendové jazyky, ako napríklad PHP alebo Node.js. Vyžiadané stránky sa pred odoslaním klientovi spracujú a upravia podľa užívateľských požiadaviek. Používa sa napríklad pri spravovaní článkov v blogoch, využitím užívateľských účtov alebo ukladaní obsahu v databázy za účelom rýchlejšej úpravy [1].

Podľa spôsobu načítavania obsahu sa delia na jedno-stránkové aplikácie a viac-stránkové aplikácie. Jednostránkové aplikácie alebo SPA používajú funkcie dynamického prepisovania obsahu, pri ktorom sa prepisuje iba konkrétny obsah, čím je načítanie stránky oveľa rýchlejšie, ako keby sa načítavala celá stránka znova. Výhodou takýchto stránok je rýchle načítanie obsahu pri prepínaní stránok alebo možnosť načítať obsah aj v režime offline. Toto je možné vďaka tomu, že celá aplikácia sa stiahne s prvotnou požiadavkou. Naopak viac-stránkové aplikácie pracujú tak, že pri zmene stránky sa vždy načíta celá stránka odznova. Tento princíp sa používa hlavne pri stránkach s veľkým množstvom obsahu. Jednostránkové aplikácie oproti viac-stránkovým sú horšími kandidátmi pri SEO optimalizáciách, a to práve tým, že obsah pri prvotnom načítaní nie je vždy dostupný [5].

2.2 Klientska časť - frontend

Frontend je časť webovej aplikácie, ktorá sa zaoberá užívateľským rozhraním a skúsenosťou. Medzi frontend sa často radí grafický dizajn, rozloženie stránky alebo funkcionalita, s ktorou užívateľ priamo interaguje či už sú to rôzne animácie alebo otváracie navigačné panely. Pri frontende sa typicky využíva HTML pre vytváranie štruktúry stránky, CSS pre vizuálnu úpravu stránky a JavaScript pre rôzne interakcie s užívateľom [13]. Frontend je veľmi dôležitá časť každej webovej aplikácie nakoľko je to práve tá reprezentatívna časť stránky. Vyžaduje pochopenie dizajnových princípov a užívateľov, ktorí túto webovú aplikáciu budú navštevovať. Preto je veľmi dôležité zistiť, pre akú cieľovú kategóriu užívateľov je daná stránka zameraná. v dnešnej dobe je pravidlom vytvárania moderných a responzívnych webových aplikácií, ktoré budú fungovať na každom zariadení či už je to mobilný telefón, tablet alebo počítač.

2.2.1 HTML

HTML[1] je hypertextový značkovací jazyk, za pomoci ktorého sa vytvárajú dokumenty pre webové aplikácie. HTML poskytuje iba štruktúru dokumentu, preto sa nepovažuje za programovací jazyk. Obsahuje veľké množstvo značiek, ktoré opisujú jednotlivé komponenty dokumentu, ako napríklad hlavička, pätička, paragraf, tabuľka alebo zoznamy.

V HTML sa značka označuje na začiatku pomocou `<` a na konci s `>`. Väčšina značiek sa vyskytuje v pároch, ktoré obklopujú obsah alebo iné podriadené značky. Príklad takýchto značiek je `<p>`, ktorá indikuje, že text má byť zobrazený ako paragraf a `</p>` označuje koniec paragrafu. Medzi týmito značkami sa nachádza obsah, ktorý bude v prehliadači zobrazený ako paragraf. Existujú však aj prázdne značky alebo nepárne značky, ktoré nepotrebujú ukončenie, pretože neobsahujú žiadny text a ani iné podriadené značky. Medzi takéto značky patrí napríklad `<hr>`, ktorá hovorí, aby prehliadač vložil do dokumentu oddelovaciu čiaru [13].

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Titulok</title>
</head>
<body>

    <h1 id="hlavny-nadpis">Nadpis</h1>
    <p>
        Lorem ipsum dolor, sit amet consectetur adipisicing elit.
        Iste numquam nesciunt, ea delectus sequi fugiat, tempora
        minima dignissimos omnis magnam saepe quaerat. Placeat
        repellat nesciunt recusandae molestias atque asperiores
        voluptatem.
    </p>

</body>
</html>
```

Obr. 2.1: Ukážka štruktúry dokumentu HTML.

[1] https://html.spec.whatwg.org/multipage

HTML značky taktiež obsahujú rôzne atribúty zapisované v začiatočných značkách. Nachádzajú sa medzi názvom značky a koncovým znakom `>`. Tieto atribúty upresňujú význam značky.

Každý dokument by mal obsahovať štyri požadované značky `html`, `title`, `head` a `body`. Značka `html` oznamuje prehliadaču, kde začína a končí zdrojový text zobrazovanej stránky. Hlavička `head` znázorňuje hlavičku dokumentu. Táto hlavička sa na stránke nezobrazuje, ale obsahuje dôležité informácie o dokumente ako napríklad názov stránky, popis, kľúčové slová, ale aj prepojenie HTML dokumentu s kaskádovými štýlmi CSS. Telo dokumentu `<body>` je obsah stránky, ktorý sa zobrazuje užívateľovi [11].

2.2.2 CSS

Kaskádové štýly alebo CSS[2] opisujú ako bude HTML dokument vyzerať. Aj napriek tomu, že každý prehliadač má preddefinované štýly pre samotné HTML značky, sa využíva CSS, aby stránky boli intuitívnejšie a príťažlivejšie. v CSS sa nastavuje napríklad farba pozadia, veľkosť a farba písma, celkový vzhľad rozloženia a iné. Pomocou CSS sa taktiež prispôsobuje ako stránka bude vyzerať na zariadeniach s rôznymi veľkosťami displeja [13].

Základná štruktúra CSS obsahuje pravidlá pre jednotlivé značky. Každé pravidlo sa skladá z dvoch hlavných častí selektor a deklarácia. Selektor udáva, na akú značku bude daný štýl použitý a deklarácia hovorí, aké štýly sa budú na daný selektor vzťahovať. Deklarácia obsahuje niekoľko rôznych štýlov, kde štýl je definovaný ako vlastnosť napríklad farba textu a jeho hodnota napríklad modrá.

```
1  * {
2      box-sizing: border-box;
3  }
4
5  html, body {
6      margin: 0;
7      padding: 0;
8      height: 100%;
9  }
10
11 #hlavny-nadpis {
12     font-size: 2rem;
13     color: #2563eb;
14 }
```

Obr. 2.2: Ukážka zápisu kaskádových štýlov, CSS.

Najčastejšie selektory môžu byť samotná HTML značka, identifikátor alebo trieda. Na rozdiel od HTML značky sa identifikátor značí s prefixom `#` a v HTML pridaním do značky atribút `id`, kde hodnota atribútu je názov identifikátora bez mriežky. Pre triedu je podobný zápis, ale namiesto mriežky sa značí s prefixom `.` a v HTML dodaním atribútu `class` [3].

[2] https://www.w3.org/Style/CSS

2.2.3 JavaScript

JavaScript[3] je typ skriptovacieho jazyka, ktorý sa používa pre pridávanie rôznych animácií, správania alebo priamych interakcií s používateľom. Môže byť používaný taktiež aj pre manipuláciu s objektovým modelom dokumentu (DOM), ktorou je možné meniť štruktúru a obsah stránky [13]. Taktiež je možné využiť JavaScript pre posielanie asynchrónnych požiadavkou na server, vďaka ktorým je možné dynamicky manipulovať s webovou aplikáciou bez nutnosti obnovenia aktuálne otvorenej stránky.

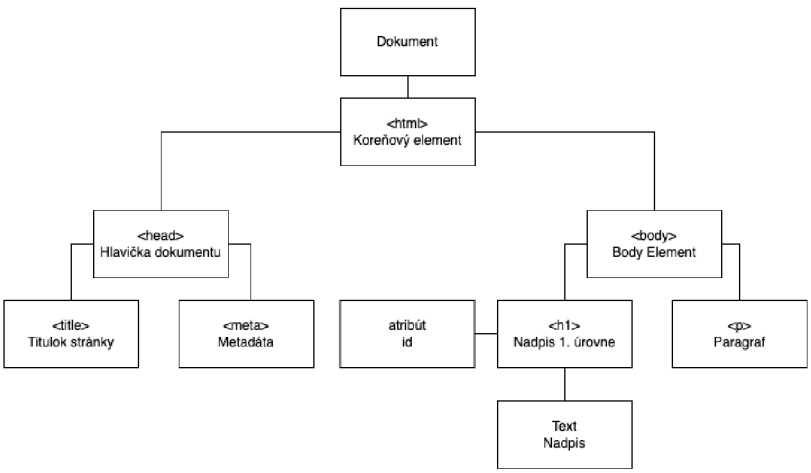
Základné prvky programovania v jazyku JavaScript sú premenné a funkcie. Premenné môžu byť globálne, dostupné v celom kóde alebo lokálne, dostupné iba v rámci určitej funkcie, kde boli definované. Základné dátové typy ako číslo, reťazec, bool, undefined a null sa predávajú pri volaní funkcií hodnotou, takže pôvodná hodnota premennej sa nezmení, ale skopíruje sa a následne sa s ňou pracuje vo funkcií. Naopak zložitejšie dátové typy, ako napríklad pole alebo objekt sa predávajú odkazom, čo znamená, že všetky zmeny v premennej vo funkcií sa aplikujú aj mimo funkcie, ktorú sme volali s parametrom danej premennej [16].

```
1  const text = "Text priradený k premennej";
2
3  function updateHeading(text) {
4      document.querySelector('#hlavny-nadpis').innerHTML = text;
5  }
6
7  updateHeading(text);
```

Obr. 2.3: Ukážka krátkeho kódu v jazyku JavaScript.

Document Object Model (DOM)

Document Object Model[4] je zoznam štandarizovaných elementov HTML dokumentu. Je veľmi často spájaným s JavaScriptom práve kvôli možnostiam manipulácie jednotlivých elementov v dokumente [13].



Obr. 2.4: Ukážková štruktúra DOM.

[3] https://developer.mozilla.org/en-US/docs/Web/JavaScript
[4] https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

2.3 Serverová časť - backend

Backend je časť webovej aplikácie, ktorá sa zaoberá spravovaním požiadavkou, ktoré odoslal klient serveru. Jednotlivé požiadavky sa spracujú a odošlú odpoveď. Táto odpoveď môže závisieť od parametrov, ktoré daná požiadavka má. Odpovede vždy obsahujú kód (HTTP Status Code) [9], ktorý udáva, aký typ odpovede bol obdržaný zo servera. Taktiež sa backend stará o komunikáciu s databázou, z ktorej si získava informácie pomocou príkazov.

2.3.1 PHP

PHP[5] je skriptovací jazyk využívaný v backende webových aplikácií, pre vytváranie dynamických stránok. Dynamické stránky je možno vytvárať vďaka možnosti vloženia PHP kódu do HTML dokumentu. z vloženého kódu, sa pri každom načítaní stránky vygeneruje HTML alebo iný výstup posielaný klientovi.

Každý PHP kód, sa začína pomocou otváracieho znaku `<?php` a ukončuje sa pomocou `?>`. Súbor, ktorý obsahuje iba PHP nie je nutné ukončovať. Taktiež je možné využiť skrátenú formu pre výstup pomocou otváracieho znaku `<?=`.

```
1  <?php
2
3  $heading = "Hlavný nadpis";
4
5  ?>
6
7  <h1><?= $heading ?></h1>
```

Obr. 2.5: Ukážka krátkeho PHP kódu s využitím HTML.

Integrácia databáz v PHP je natívna, kde je možné priamo pripojiť databázový server s backendom. Obsahuje taktiež triedu PHP Database Objects (PDO), ktorá priamo pracuje s databázou pomocou overeného bezpečného prístupu [15].

Od verzie 5, PHP taktiež podporuje objektovo orientované programovanie, ktoré sa dodnes využíva hlavne pri frameworkoch s využitím softwarovej architektúry Model-View-Controlle.

2.3.2 MySQL (Databáza)

MySQL[6] je relačný databázový systém. Databáza dokáže pridávať, zobrazovať, upravovať, vymazávať alebo filtrovať dáta podľa potreby. Systém je schopný pracovať s viacerými užívateľmi a požiadavkami naraz. Využíva štandarizovaný štrukturovaný dopytovací jazyk (SQL), pomocou ktorého je možné posielať príkazy na databázový server. Dôvodom používania databáz oproti súborom je hlavne rýchlosť získavania dát alebo využívanie dotazov, ktoré dokážu získať presné dáta podľá kritérií. Typ relačná databáza je najbežnejší typ databáz, pri ktorých tabuľky majú určitú štruktúru, podľa ktorej sa dáta ukladajú. Jednotlivé tabuľky obsahujú stĺpce, ktoré definujú názov, typ dát, ktoré stĺpec obsahuje, a rôzne

[5] https://www.php.net/
[6] https://www.mysql.com/

atribúty, kľúče, indexy a pod. Každý riadok v tabuľke obsahuje práve jeden záznam štrukturovaný podľa stĺpcov. v prípade vzťahov s inými tabuľkami je možné použiť cudzie kľúče. Tieto kľúče jednoznačne identifikujú vzťah medzi dvoma tabuľkami [15].

2.4 REST API

REST API je typ rozhrania, ktoré poskytuje informácie pri komunikácií server-klient pomocou protokolu HTTP. Dáta môžu byť odosielané vo formáte JSON, HTML, XLT, PHP alebo obyčajný text [12]. Využíva štandardné HTTP metódy ako sú `GET`, `POST`, `PUT` a `DELETE`, ktorými sa reprezentuje, aká operácia sa má vykonať. Hlavnou výhodou je, že nie je nutné využívať ten istý jazyk pri frontende a backende, čo vytvára vývojárom možnosti použitia REST API v iných aplikáciách.

Kritéria, ktoré považujú API za REST API definované v [12] sú:

- *Architektúra klient-server s požiadavkami zasielanými pomocou HTTP*
- *Bezstavová komunikácia, pri ktorej nie je uchovaná žiadna informácia klienta medzi požiadavkami a každý požiadavok je separátny*
- *Možnosť cachovania dát*
- *Jednotné rozhranie medzi komponentmi, takže informácie sú prenášané v štandardnej forme*
- *Vrstvený systém, ktorý organizuje každý typ servera (zodpovedné napríklad pri bezpečnosti), aby zahŕňal získavanie požadovaných informácií do hierarchií, ktoré klient nevidí*
- *Kód na vyžiadanie (voliteľné), možnosť odoslať spustiteľný kód zo servera na klient.*

V praktickej časti bakalárskej práce je používaná práve komunikácia REST API, kde klient posiela požiadavky serveru pomocou HTTP klienta axios[7], s využitím štandardných HTTP metód, ktoré server následne spracuje a odpovie s korenšpondujúcou odpoveďou.

2.5 Využité technológie pri implementácií

Spomenuté technológie, ktoré boli použité pri praktickej časti, napomáhajú spojiť frontend a backend do jednej celistvej aplikácie. Pri Reacte sa používajú stavy, hook-y, prípadne manažovanie globálnych stavov, aby bolo možné jednotlivé prvky, ktoré boli prijaté z backendu, využiť jednoducho vo viacerých komponentoch. Následne prepojenie aplikácií s pomocou WebSocketom, umožní priame pripojenie medzi užívateľom a serverom bez prerušenia, čo napomáha pri vzdialenej kooperácií.

[7] https://axios-http.com/

2.5.1 React.js

React.js[8] je JavaScriptová frontendová knižnica vyvinutá firmou Facebook. Pomocou tejto knižnice sa vytvárajú komponenty, ktoré spolu tvoria užívateľské rozhranie. Tieto komponenty sú znovupoužiteľné, čo znamená, že každý komponent je možné použiť na viacerých miestach, bez nutnosti znovu vytvárania rovnakých komponentov. React.js využíva taktiež funkcie virtual DOM, vďaka ktorým umožňujú aktualizáciu jednotlivých komponentov a nie je nutné znova vykresľovať celé užívateľské rozhranie, ale iba daný komponent [7].

Komponenty

Jednotlivé komponenty sa píšu pomocou rozšíreného syntaxu jazyka JavaScript nazývaný JSX. Ide o klasický JavaScript, ktorý podporuje písanie HTML elementov priamo v jazyku JavaScript. Komponent môže byť trieda alebo funkcia, v ktorej môžu byť využívané napríklad rôzne stavy, definícia funkcií pre ovládanie komponentu a udalostí, alebo hooks. Tento komponent potom vracia HTML elementy, ktoré popisujú jeho štruktúru. Komponent musí vracať vždy iba jeden rodičovský element, ktorý ale môže mať neobmedzený počet potomkov. v prípade, že nie je potrebné vykresľovať rodičovský element, je možné použiť komponent `Fragment` alebo obalenie komponentu do prázdneho elementu `<></>`. Vytváranie triednych komponentov je v dnešnej dobe už menej využívané a taktiež to nie je odporúčané v oficiálnej dokumentácií[9] React-u. Funkčné komponenty sú viac presadzované hlavne kvôli ich ľahšej čitateľnosti a pochopenia, sú rýchlejšie pri vykreslovaní a podporujú hook-y, vďaka ktorým sa práve prešlo z triedneho štýlu na funkčný typ definovania komponentov [7].

Stavy

Stavy sú objekty komponentu, ktoré uchovávajú dáta, na základe, ktorých sa obsah aktualizuje. Stav môže byť napríklad počítadlo klikov, kde sa na tlačítko nastaví udalosť pri kliknutí, aby sa inkrementoval počet kliknutí o 1. Vďaka stavom môžu byť komponenty dynamické a reagujúce na užívateľské vstupy a iné udalosti. Každou aktualizáciou stavu sa komponent znova vykreslí s už aktualizovanou hodnotou [7].

Hooks

Hooks sú funkcie, ktoré umožňujú funkčným komponentom využívať práve spomínané stavy alebo iné vlastnosti, ktoré boli pôvodne umožnené v komponente definované skrz triedy. Práve vďaka ním sa začali písať komponenty ako funkcie od React.js verzie 16.8. Hooks je možné taktiež vytvárať vlastné, čím umožňujú byť jednotlivé komponenty viac flexibilné a ľahko spravovateľné. React poskytuje viacero užitočných hooks. Medzi najzákladnejšie patria:

- `useState` - slúži, podľa názvu, na správu stavov. Hook vracia dve hodnoty, ktorými sú samotný stav a funkciu pre prideľovanie stavu. Zvyčajne sa hodnoty nazývajú `state` a `setState`, kde state je názov stavu, ktorým môže byť napríklad "counter"(počítadlo).

[8] https://react.dev/
[9] https://react.dev/reference/react/PureComponent#alternatives

- `useEffect` - sa využíva napríklad v prípade sťahovania dát z REST API alebo na manipulovanie DOM. Obsahuje dva parametre, ktorými sú funkcia, ktorá sa má vykonať, a pole závislostí, ktorý je voliteľný. Ak nie je zadaný tento parameter, hook sa spustí vždy, keď sa prvý a každý krát vykreslí komponent. Pokiaľ je pole prázdne vykoná sa iba pri počiatočnom vykreslení. v prípade, že toto pole bude obsahovať pole definovaných stavov, tento hook sa vykoná vždy keď sa tieto stavy zmenia.
- `useRef` - umožňuje definovať referenciu pre element, vďaka ktorej je možné ľahko získavať informácie o elemente alebo ho upravovať.
- `useContext` - povoľuje ostatným potomkom rodičovského komponentu pristupovať ku stavom, premenným a funkciám, ktoré boli definované v rodičovskom komponente. Slúži hlavne pre správu globálnych stavov, aby nebolo nutné, potomkom odovzdávať tieto stavy v atribútoch.

Manažovanie globálneho stavu

Manažovanie globálneho stavu môže byť sprostredkované vďaka spomínanému hook-u `useContext` alebo vďaka externým knižniciam a balíčkom. Jeden z veľmi používaných balíčkov je Recoil.js[10]. Pomocou tohto balíčka je možné vytvárať a manažovať jednoducho stavy, vytváraním atómov a selektorov.

Atóm je klasický stav, komponentu, ktorý môže byť prístupný vo všetkých komponentoch. Manažovanie stavu je sprostredkované pomocou hook-u `useRecoilState`, prípadne `useRecoilValue`, pre získanie iba hodnoty stavu, alebo `useSetRecoilState` pre získanie funkcie pre nastavovanie hodnoty stavu.

Selektory reprezentujú odvodené stavy. Používajú sa napríklad pri filtrovaní stavov, ktorý môže obsahovať zoznam a hodnotu, podľa ktorej chceme daný zoznam filtrovať.

Routing

Pre routovanie na frontende slúži knižnica React Router[11], vďaka ktorej je možné načítavať stránky bez nutnosti obnovenia stránky. Princíp takejto webovej aplikácie je spomínaná jednostránková aplikácia. v routeri sa musia definovať všetky možné cesty, ktoré budú dostupné a k ním priradiť komponent, ktorý sa bude vykresľovať. Cesta môže obsahovať aj dynamické parametre, ktoré sa dajú získať pomocou hook-u `useParams`. Tieto parametre sa potom použijú v prípadných sťahovaní dát pomocou REST API alebo pri zmene obsahu na základe tohto parametra.

2.5.2 Laravel

Laravel[12] je aplikačný rámec pre programovací jazyk PHP. Je to jeden z najpopulárnejších PHP frameworkov. Využíva softwarovú architektúru Model-View-Controller. Pevne sa drží svojich konvnecií, ktoré uľahčujú prácu všetkým programátorom. Postupom času boli pre Laravel vyvinuté rôzne podporné balíčky, vďaka ktorým má Laravel už dnes vlastný ekosystém. Tento ekosystém zahŕňa knižnice a možnosti od nastavenia prostredia pre rôzne frontendové knižnice, cez monitorovanie a ladenie, až po nasadenie webovej aplikácie na server.

[10] https://recoiljs.org/
[11] https://reactrouter.com/
[12] https://laravel.com/

Aplikačný rámec používa nástroj príkazového riadku s vlastnou sadou príkazov nazývaný Artisan. Obsahuje príkazy ako napríklad generovanie súborov, vymazávanie cache, plánovanie úloh, práca s frontami alebo výpis všetkých registrovaných stránok vo webovej aplikácií.

Integrovanú má aj ochranu proti Cross-site request forgery (skrátene CSRF), ktorá chráni pred odosielaním požiadaviek z nelegetímneho zdroja. Vďaka tomu môže nelegitímny zdroj zneužívať práva napríklad prihlaseného užívateľa na danej doméne. Každý užívateľ dostane token, pomocou ktorého sa overuje, či daný užívateľ odoslal požiadavok z danej domény alebo sa jedná o podvrhnutú stránku.

Pri programovaní frontend-u priamo v aplikačnom rámci, Laravel poskytuje svoj vlastný nástroj šablón nazývaný Blade, pre tvorbu frontendu webovej aplikácie. Okrem nástrojov šablón podporuje aj knižnice ako React.js alebo Vue.js, ktoré môžu byť priamo prepojené pomocou nástroju Inertia.js[13], ktorý zabezpečuje routing, vďaka ktorému je umožnené vytvoriť viac-stránkovú aplikáciu.

```
1  <h1>
2      @if ($heading)
3          {{ $heading }}
4      @else
5          Nadpis
6  </h1>
```

Obr. 2.6: Ukážka kódu, ktorý následne Laravel kompiluje, pomocou nástroja šablón Blade.

Pre správu databáz Laravel ponúka objektovo relačné mapovanie pre modely nazývané Eloquent ORM. Táto technika zjednodušuje prácu s modelmi resp. tabuľkami, pričom dokáže s nimi jednoducho manipulovať, ako napríklad vkladať, upravovať a mazať jednotlivé záznamy z databázy alebo načítavať ich ostatné vzťahové modely. Tabuľky pre tieto modely je možné jednoducho vytvárať pomocou migrácií a použitím tovární naplňovať testovacími dátami [14].

Životný cyklus požiadavku v aplikačnom rámci Laravel obsahuje niekoľko častí. Najskôr všetky požiadavky smerujú do vstupného súboru `public/index.php`, ktorý slúži ako začiatočný bod načítania celého aplikačného rámca. Prichádzajúca požiadavka je následne zaslaná HTTP jadru. Toto HTTP jadro nakonfiguruje všetky potrebné funkcie Laravelu, a vykoná všetky potrebné úlohy predtým, ako je požiadavka spracovaná. Definuje aj zoznam midlvérov, cez ktoré musia požiadavky prejsť predtým, ako sú spracované. Midlvéry sa starajú napríklad o to, aby požiadavka bola zamietnutá v prípade, že aplikácia je v móde údržby alebo neoverí správne CSRF token. Jedna z veľmi dôležitých častí je poskytovateľ služby. Poskytovateľ služby zodpovedá za načítanie komponentov, ako napríklad databáza, fronty, validácia a iné. Na koniec celého procesu načítania potrebných súborov a funkcií, sa spracuje požiadavka a odošle sa príslušná odpoveď klientovi.

Aplikačný rámec

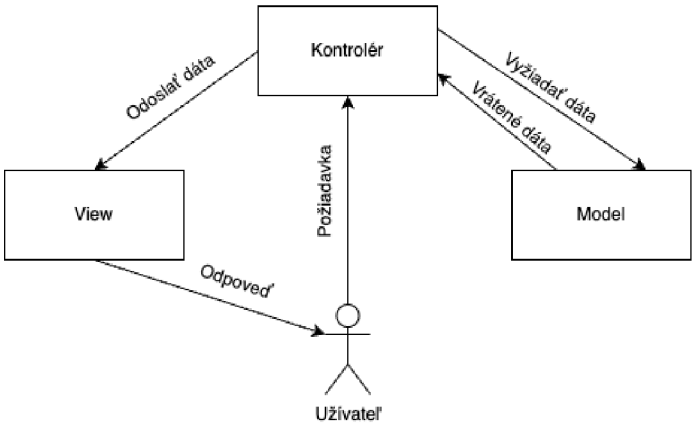
Aplikačný rámec je podporná štruktúra pri programovaní, ktorá obsahuje preddefinované funkcie, ktoré sa používajú pri programovaní webových alebo aj iných aplikácií. Obsa-

[13] https://inertiajs.com/

huje taktiež aj odporúčanú štruktúru, knižnice, komponenty a osvedčené postupy, ktorými sa proces vyvíjania zrýchľuje.

Model-View-Controller (MVC)

Model-View-Controller skrátene MVC, je softwarová architektúra, ktorá rozdeľuje celú logiku do troch navzájom prepojených častí model, view a kontrolér. Model má na starosti prácu s dátami, ich manipuláciu a získavanie. View je frontendová časť, ktorá zvyčajne reprezentuje HTML, CSS a JavaScript. Kontrolér sa správa ako sprostredkovateľ medzi modelom a view. Spracováva požiadavky odoslané od klienta (view) a následne zaisťuje zmeny v modely. Tieto zmeny sú následne odoslané späť na klienta čím aktualizuje samotný view.



Obr. 2.7: Diagram, zobrazujúci spojitosť jednotlivých častí modelu MVC.

2.5.3 WebSocket

WebSocket je komunikačný protokol, ktorý zabezpečuje obojsmernú komunikáciu medzi klientom a serverom pomocou protokolu riadenia prenosu, inak nazývaného TCP. Vďaka WebSocketom je teda umožnené webovej aplikácií komunikovať so serverom v reálnom čase, čím užívateľ je schopný dostávať aktualizácie bez toho, aby si ich priamo vyžiadal od servera. Používa sa napríklad pri chatovacích aplikáciách, notifikáciach alebo aj grafických štatistikách v reálnom čase. Životný cyklus WebSocketov sa skladá z troch fáz: otvorenie spojenia, odosielanie a prijímanie dát, a ukončenie spojenia [10].

Otvorenie spojenia medzi klientom a serverom začína jednoduchou HTTP požiadavkou. Následne server odpovie a spojenie medzi klientom a serverom je vytvorené. Tento proces sa nazýva handshake. Handshake WebSocketov je definovaný dokumentom RFC6455 [6]. Hlavičky odoslané klientom pre otvorenie spojenia sú definované nasledovne:

```
GET /room HTTP/1.1
Host: example.com
Connection: Upgrade
Upgrade: websocket
Origin: http://example.com
Sec-WebSocket-Key: c2NoZW1hdGUtYXBw
Sec-WebSocket-Protocol: room
Sec-WebSocket-Version: 13
```

Obr. 2.8: Rozdiel medzi pripojením s pomocou WebSocketov a štandardným HTTP pripojením.

Hlavičky, ktoré sa posielajú s prvou HTTP požiadavkou od klienta k serveru obsahujú pripojenie (`Connection`) typu `Upgrade`. Hlavička `Upgrade` sa používa od HTTP 1.1, ktorá slúži pre aktualizáciu už vytvoreného pripojenia medzi klientom a serverom na iný protokol. Ide napríklad o aktualizáciu protokolu HTTP a HTTPS na spomínaný WebSocket. Pomocou `Sec-WebSocket-Key` je časť informácie, ktorou dokazujeme serveru, že sa jedná o validný WebSocket požiadavok. `Sec-WebSocket-Protocol` určuje jeden alebo viac pod protokolov, ktoré sú definované v [8]. Verziu WebSocketu určuje `Sec-WebSocket-Version`, vďaka ktorej server správne interpretuje otvorenie spojenia, prenos dát a ukončenie spojenia. Server po spracovaní požiadavky odpovedá pomocou nasledujúcich hlavičiek:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: YmFjaGVsb3ItdGhlc2lz
Sec-WebSocket-Protocol: room
```

V odpovedi sa odosiela hlavička `Sec-WebSocket-Accept`, ktorá je zložená zo spomenuhého kľúču `Sec-WebSocket-Key` ku ktorému bol priradený globálny unikátny identifikátor inak nazývaný GUID[14] a nakoniec celé enkódované do base64[15].

Redis

Redis je databázový systém, fungujúci na princípe kľúč-hodnota, čo znamená, že všetky hodnoty, sú reprezentované ako dvojica, kde kľúč slúži ako identifikátor, podľa ktorého sa daná hodnota vyhľadáva. Tento systém uchováva všetky tieto dvojice v operačnej pamäti servera, čím zabezpečí vysokú rýchlosť načítavania dát. Používa sa ako databáza, úložisko cache alebo sprostredkovateľ správ. Podporuje dátové typy ako napríklad klasický reťazec, hash, zoznamy alebo sety [4].

V tejto práci je Redis používaný práve pre sprostredkovanie správ, pri WebSocketoch, kde server ukladá užívateľov, ktorí sú pripojení v jednotlivých schémach, aby bolo možné zistiť koľko a akí ľudia sú momentálne pripojení v schéme pri kooperácií. Pre spojenie s týmto serverom sa používa balíček `Laravel Echo Server`[16], ktorý následne posiela na Redis dáta, ktoré má uložiť. Na frontende je prepojenie s Laravel Echo Server zabezpečené

[14] https://www.rfc-editor.org/rfc/rfc4122
[15] https://www.rfc-editor.org/rfc/rfc4648
[16] https://github.com/tlaverdure/laravel-echo-server

pomocou Socket.io[17] a Laravel Echo[18]. Tieto dáta sa posielajú iba v prípade, že sa využíva kanál prítomností, pri ktorom je možné užívateľov zaznamenávať.

[17] https://socket.io/
[18] https://github.com/laravel/echo

Kapitola 3

Analýza a porovnanie existujúcich aplikácií

V tejto kapitole sa spomínajú kritéria, ktoré sú vyžadované pri praktickej časti bakalárskej práce. Ďalej sa porovnávajú už existujúce aplikácie, ktoré pracujú podobne. Zaoberá sa ich nedostatkami, ktoré nespĺňajú požadované kritéria. v neposlednom rade popisuje a navrhuje, akým spôsobom sa tieto nedostatky riešia v danej praktickej časti. Kritéria sú následne zobrazené v tabuľke s porovnávajúcimi aplikáciami. Tabuľka obsahuje či dané kritérium bolo splnené úplne alebo len čiastočne.

3.1 Kritéria webovej aplikácie

Medzi hlavné kritéria pre túto webovú aplikáciu patria:

1. jednoduchosť a intuitívnosť užívateľského rozhrania
2. možnosť vytvárať návrh databázových schém pomocou pseudo-kódu a vstupných polí
3. exportovanie do migračných súborov a modelov pre aplikáciu Laravel
4. umožnenie vzdialenej kooperácie pri vytváraní tabuliek

Prvé dôležité kritérium hovorí o tom, ako je v dnešnej dobe veľmi dôležitá jednoduchosť a intuitívnosť užívateľských rozhraní. Veľa aplikácií je prispôsobených práve pre určitú kategóriu užívateľov, ktorí danú aplikáciu budú navštevovať, preto je nutné prispôsobiť zobrazený obsah tak, aby sa v ňom vedeli rýchlo zorientovať.

Druhé kritérium hovorí o jednom zo základných funkcionalít, ktoré je vyžadované touto webovou aplikáciou. Aplikácia je priamo zameraná na možnosť vytvárať databázové schémy pomocou pseudo-kódu alebo vstupných polí. Pseudo-kód predstavuje rýchlejšiu formou vytvárania týchto schém nakoľko nie je nutnosť používať obe zariadenia t.j. klávesnicu a myš, ale stačí používať iba klávesnicu. Vstupné polia predstavujú síce pomalšiu formu, ale aj napriek tomu existuje veľa ľudí, ktorí preferujú vstupné polia namiesto písania kódu.

Tretím kritériom je spomínaný export migračných súborov a modelov pre aplikáciu Laravel. Tento export je nutnosť, nakoľko celkový cieľ tejto práce je eliminovať čas medzi návrhom diagramov v externom programe a písaním migračných súborov a modelov v aplikácií Laravel.

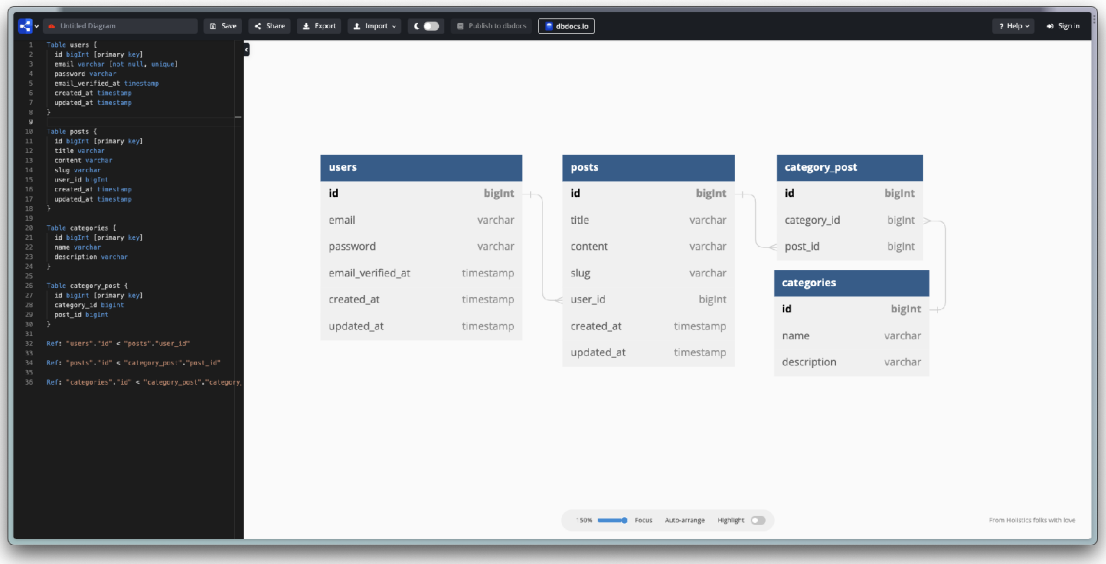
Posledné hlavné kritérium popisuje schopnosť aplikácie využiť webové sokety vo svoj prospech tak, aby bola umožnená vzdialená kooperácia. Táto kooperácia pomáha ľuďom komunikovať a riešiť zadané problémy aj napriek tomu, že nie sú priamo vedľa seba.

3.2 Podobné webové aplikácie

Existuje mnoho podobných aplikácií, ale nie každá spĺňa kritéria praktickej časti tejto bakalárskej práce. Analýza sa týka troch hlavných typov aplikácií, ktorými sú **dbdiagram.io**, **DrawSQL** a **diagrams.net**. Nasledujúce sekcie popisujú, kľúčové vlastnosti aplikácie a akým štýlom, sa v danej aplikácií vytvárajú databázové schémy.

dbdiagram.io

dbdiagram.io[1] je kvalitná webová aplikácia na tvorenie diagramov pomocou vlastného pseudo-kódu nazývaného Database Markup Language[2] (DBML). Zvláda import a export pre rôzne databázové systémy MySQL, SQL Server alebo PostgreSQL. v reálnom čase vykresluje následne schému databázy po jednotlivých tabuľkách. Možnosť vzidalenej kooperácie je možný, ale iba po dobu, že druhá strana nebude mať povolené upravovať kód, v opačnom prípade je nutné zakúpenie prémia.

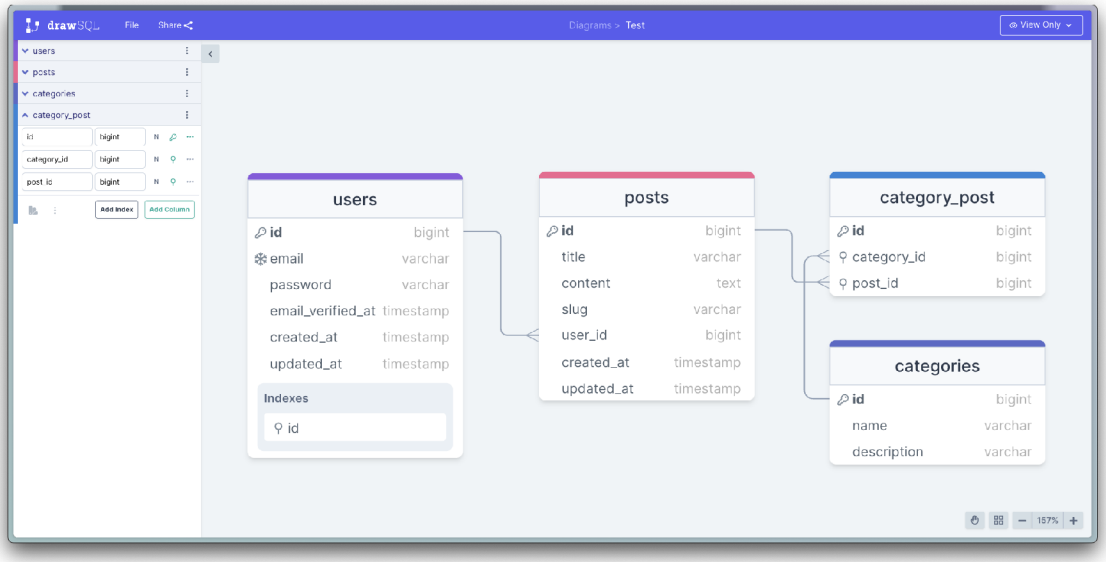


Obr. 3.1: Webová aplikácia dbdiagram.io, ktorá generuje databázové schémy pomocou pseudo-kódu DBML.

[1] https://dbdiagram.io/
[2] https://dbdiagram.io/docs/dbml

DrawSQL

DrawSQL[3] je ďalšia skvelá aplikácia na vytváranie databázových schém. Podobnosťou sa približuje najviac k tejto bakalárskej práci. Dokáže exportovať migračné súbory pre Laravel alebo špeciálne súbory pre Laravel Blueprint[4]. Laravel Blueprint je aplikácia pre príkazový riadok Artisan, ktorá pomocou príkazov je následne schopná vytvoriť modely, a k ním aj migračné súbory. DrawSQL poskytuje intuitívne užívateľské rozhranie, kde je umožnené vytvárať jednotlivé tabuľky pomocou vstupných polí. Je možné využiť vzdialenú kooperáciu taktiež a aj s úpravu, ale iba v prípade, že osoba patrí do tímu s osobou, ktorá poskytla odkaz na kooperáciu.

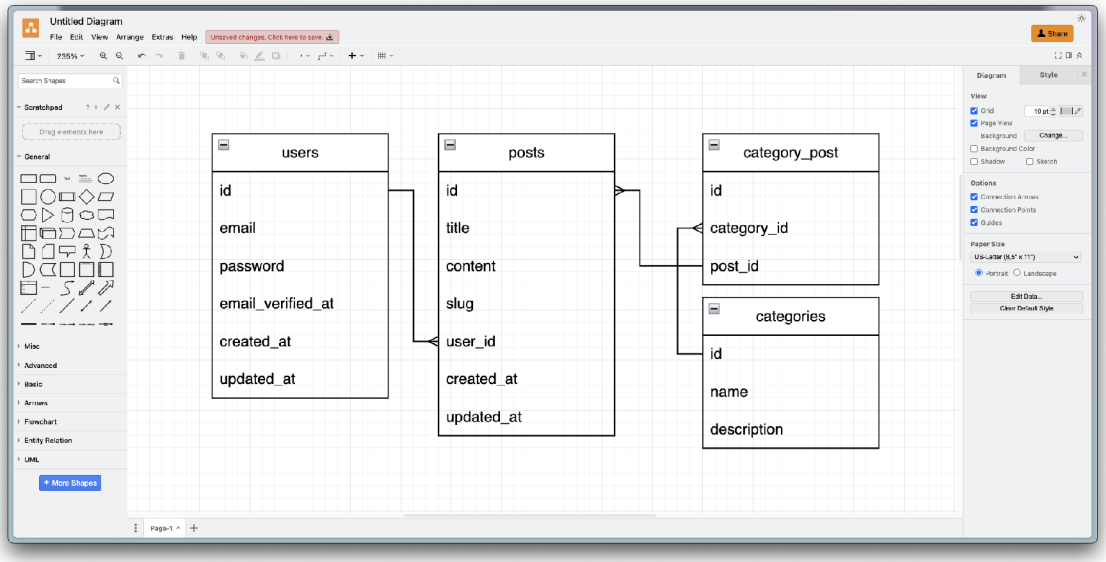


Obr. 3.2: Webová aplikácia DrawSQL, ktorá generuje databázové schémy pomocou vypľňovacích polí.

[3] https://drawsql.app/
[4] https://blueprint.laravelshift.com/

diagrams.net

diagrams.net[5] je posledná zo spomínaných aplikácií. Táto aplikácia nie je primárne určená iba na tvorbu databázových schém. Používa sa taktiež na vytváranie iných diagramov, ako napríklad diagram prípadov použitia, entitne vzťahový model a pod. Funguje za pomoci drag and drop editora, do ktorého sa jednoducho presúvajú potrebné elementy. Nakoľko nástroj nie je primárne určený pre tvorenie štruktúr databáz, je práve najviac vzdialený od podobnosti k tejto bakalárskej práci.



Obr. 3.3: Webová aplikácia diagrams.net, pomocou ktorej sa vytvárajú diagramy pomocou drag and drop editora.

3.3 Zhrnutie výsledkov

Všetky tri aplikácie majú prívetivé užívateľské prostredie, v ktorom sa dá ľahko orientovať. Hneď z prvého pohľadu je zreteľné, čoho je aplikácia schopná a aké sú ich možnosti. Na prvý pohľad má však aplikácia DrawSQL niektoré funkcie jemne mätúce, ako napríklad zadávanie stĺpcov tabuliek. Jednotlivé dátové typy tabuliek sú dostupné až po prvom zadanom písmene, čo je nepraktické z hľadiska efektivity, nakoľko užívateľ musí zisťovať písaním písmen, aké všetky možné typy existujú. Taktiež obsahuje zvláštne ikonky pri zadávaní unikátneho kľúča, ktorý je identifikovaný ako snehová vločka, čo na prvý pohľad môže byť nezrozumiteľné. Pri možnosti vymazania stĺpca je potrebné kliknúť dva krát, čo taktiež je veľmi nepraktické. Aplikácie dbdiagram.io a diagrams.net sú veľmi prehľadné a počas analýzy neboli nájdené žiadne problémy pri užívateľskom rozhraní. Aj napriek tomu, že diagrams.net je veľmi robustná aplikácia, všetky funkcie sú ľahko dostupné a nájditeľné.

Každá spomenutá aplikácia je taktiež schopná vytvárať databázové schémy, avšak ani jedna nespĺňa kritéria, ktoré vyžadujú schopnosť vytvárania schém pomocou oboch prístupov pseudo-kódu a vstupných polí. Posledná aplikácia diagrams.net nesplňuje toto kritérium ani v jednej z možností, nakoľko vytváranie schém je vykonané pomocou editora drag and

[5] https://diagrams.net/

drop, kde sa priamo vkladajú diagramy z navigácie. Aplikácia je, ale naopak, veľmi obsiahla a dokáže vytvoriť množstvo rôznych diagramov, ktoré sa využívajú v každodennom živote. Aplikácia dbdiagram.io, ktorá používa vlastný pseudo-kód, môže prísť užívateľovi na prvý pohľad trochu mätúca svojím syntaxom. Syntax je obšírnejší a ťažšie zapamätateľný, no napriek tomu spĺňa svoju funkcionalitu. Pseudo-kód sa píše do editora Monaco[6], ktorý je vyvinutý spoločnosťou Microsoft. Tento editor je taktiež známy pod menom Visual Studio Code[7] ako desktopová aplikácia. Editor je implementovaný so základnými ale aj pokročilými funkciámi, ktoré sú dostupné pomocou príkazovej palety.

Vyžadovaný export pre aplikačný rámec Laravel z týchto troch aplikácií splňuje iba druhá aplikácia, DrawSQL. Táto aplikácia dokáže exportovať priamo migračné súbory alebo súbory pre Laravel Blueprint, pomocou ktorého sa vygenerujú modely a migračné súbory. Prvá aplikácia dbdiagrams.io napriek tomu poskytuje export pre viaceré typy, ako napríklad MySQL, PostreSQL alebo PDF dokument. Taktiež ponúka importovanie vytvorených schém z MySQL, PostgreSQL alebo aj Rails, ktorá je dostupná pre aplikačný rámec Ruby on Rails[8].

Vzdialená kooperácia je umožnená pri všetkých troch aplikáciách, avšak pri aplikácií DrawSQL je v základnej verzií možné len staticky zobraziť výsledný diagram. Po úprave diagramu zmeny nastanú u druhého užívateľa až po obnovení stránky. Funkcionalita je pridaná po zakúpení prémia. Pri aplikácií dbdiagram.io, je táto funkcionalita podobná a taktiež platená. Diagrams.net ponúka vzdialenú kooperáciu zadarmo pomocou Google Drive.

Tabuľka 3.1: Analýza kritérií, definované v 3.1, podobných webových aplikácií

Číslo kritéria	dbdiagram.io	DrawSQL	diagrams.net
1	100%	80%	100%
2	Iba pseudo-kód	Iba vstupné polia	Nie
3	Nie	Áno	Nie
4	Áno (platené)	Áno (platené)	Áno

[6] https://microsoft.github.io/monaco-editor/
[7] https://code.visualstudio.com/
[8] https://rubyonrails.org/

Kapitola 4

Návrh aplikácie

Podľa analýzy jednotlivých webových stránok, boli pripravené návrhy. Návrhy riešenia postupujú podľa navrhnutých kritérií, ktoré boli vybrané podľa dôležitosti, ktorá sa vyžaduje v danej webovej aplikácií. Návrh sa začal užívateľským prostredím, po ktorom nasledovala základná funkcionalita, ktorou je vytváranie databázových schém. Ďalší návrh zahrňuje funkcionalitu exportu vytvorenej schémy do migračných súborov a modelov. v neposlednom rade sa zaoberá využitím webových soketov pre vzdialenú kooperáciu viacerých užívateľov.

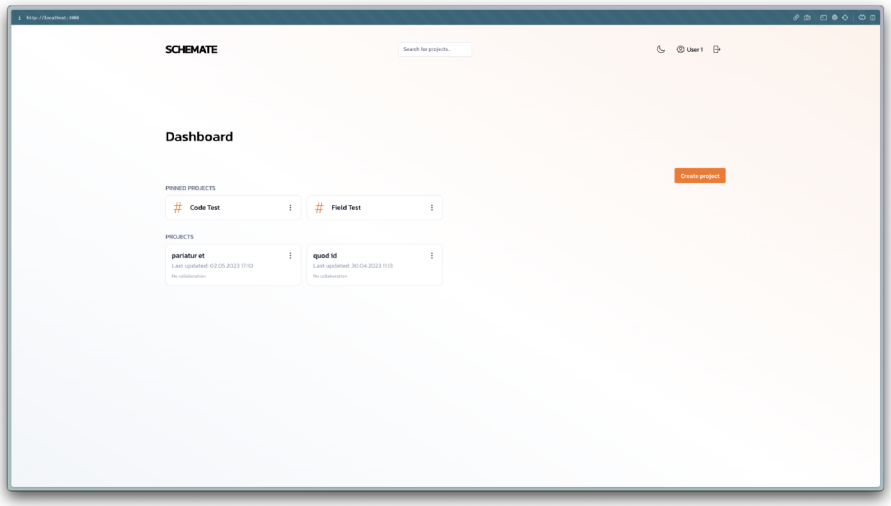
4.1 Užívateľské rozhranie

Prvým dôležitým bodom je návrh, teda užívateľské rozhranie a správne navrhnutie rôznych interakcií s užívateľom. Vysoká dôraznosť sa kladie na tento bod hlavne z dôvodu, že užívateľské rozhranie je prvé, čo užívateľ vidí ako výsledok práce. Zle navrhnuté užívateľské prostredie môže viesť k tomu, že užívateľ môže stratiť záujem o produkt. Jednoduchosť a intuitívnosť prostredia ovplyvňuje taktiež produktivitu tým, že užívateľ veľmi rýchlo pochopí, ako aplikácia funguje a aké správanie môže užívateľ očakávať. Užívateľská skúsenosť sa týka hlavne dojmu, pri využívaní rôznych interakcií s webovou aplikáciou alebo produktom. v dnešnej dobe je zaužívané, aby užívateľ dostal spätnú väzbu, ak klikne alebo položí kurzor na určitý objekt. Platí ale, že príliš veľa interakcií môže mať negatívny a kontraproduktívny vplyv na užívateľskú skúsenosť. Hlavným dôvodom je, že užívateľ je príliš rozptýlený rôznymi efektami a interakciami, ktoré zapríčiňujú stratu sústredenia.

Určenie správnej cieľovej kategórie, ktorá bude využívať danú webovú aplikáciu, je taktiež veľmi dôležité. Cieľová kategória pre túto webovú aplikáciu sú hlavne programátori a návrhári databázových schém, takže ľudia, ktorí by mali vedieť základné princípy programovania a funkčnosť databáz. Pre programátorov, je preto umožnené vytvárať tieto schémy pomocou navrhnutého pseudo-kódu. Pseudokód má veľmi jednoduchú syntax a vďaka príkazom a niektorým vstavaným funkciám dokáže pripraviť databázu pre každého. Jednoduchší postup pre užívateľov, ktorí skôr navrhujú databázové schémy, je umožnené vytvárať tieto schémy pomocou zadávacích vstupov.

Rozhranie má jednoduchý vzhľad, bez nadmerne výrazných farieb. Namiesto toho sú aplikované jemné odtiene hlavných farieb. Využíva sa veľmi populárne dizajnérske pravidlo 60-30-10. Toto pravidlo sa uplatňuje ako pri interiéroch [2], tak aj pri dizajne webových a iných aplikácií. Pravidlo hovorí o troch farbách a ich percentuálny podiel v dizajne. 60% je hlavná, dominantná farba, ktorá sa typicky používa na tie najväčšie plochy, ktorými sú napríklad pozadie stránky. 30% určuje sekundárnu farbu, ktorá sa využíva na vytvorenie

kontrastu k pozadiu. Vo webových aplikáciách to môže byť napríklad oddelenie rôznych elementov od pozadia, navigačný panel alebo pätička stránky. Posledných 10% určuje dôraz dôležitých elementov v dizajne. Využíva sa hlavne farba, ktorá upozorňuje na špecifické elementy v dizajne, tým môžu byť napríklad tlačítka alebo iné dôležité body užívateľského rozhrania.

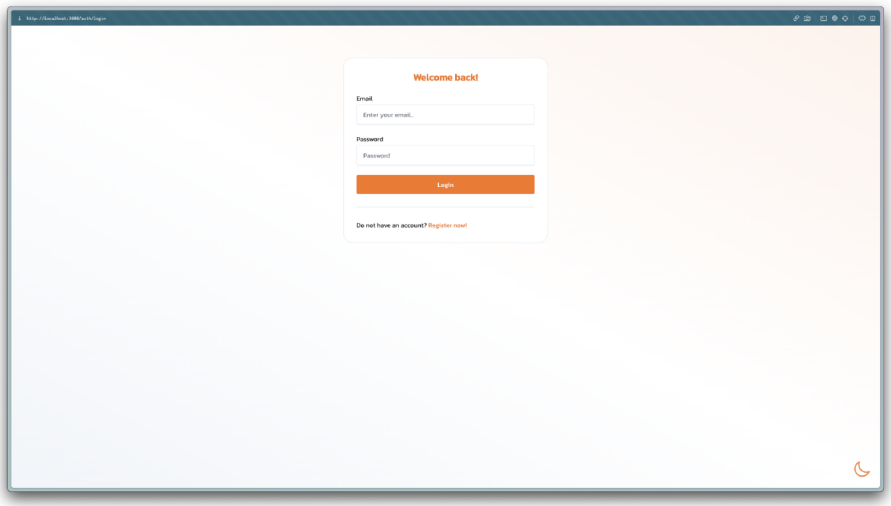


Obr. 4.1: Úvodná stránka so zoznamom projektov

Výber správnej farebnej palety je jedným z dôležitých aspektov dizajnu webových stránok. Dizajn využíva 3 farby pre každý motív. Existuje motív bledý, teda klasický a tmavý, ktorý začal byť v posledných rokoch čím viac populárny. Bledý motív využíva bielu farbu a pre dodanie kontrastu, jemnú šedú. Pri tmavom motíve sa naopak používa čierna a jej jemnejší odtieň. Ako farba, na ktorú je kladený dôraz, bola vybraná oranžová.

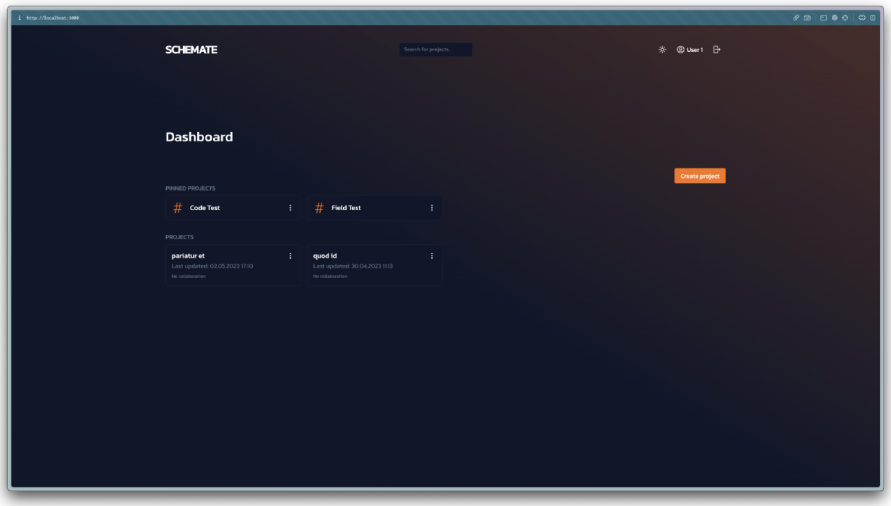
Obr. 4.2: Výber farebnej palety pre tmavý a bledý režim.

Webová aplikácia obsahuje tri dôležité stránky a ich rozloženia. Prvým je prihlásenie a registrácia užívateľa. Stránka obsahuje klasický formulár prihlásenia, prípadne registrácie, kde je možné vyplniť údaje a jednoducho sa prihlásiť.



Obr. 4.3: Ukážka prihlasovacej stránky

Dashboard je druhou veľmi dôležitou stránkou, ktorá obsahuje všetky vytvorené projekty resp. databázy. Zobrazuje aj databázy, ktoré sú v kooperácií s inými užívateľmi. Jednotlivé projekty sa dajú filtrovať, spravovať alebo pripnúť na vrch. z tejto stránky možno vytvoriť aj nový projekt pomocou tlačítka "Create project". Po kliknutí sa zobrazí dialóg, v ktorom bude užívateľ požiadaný o zadanie názvu projektu.

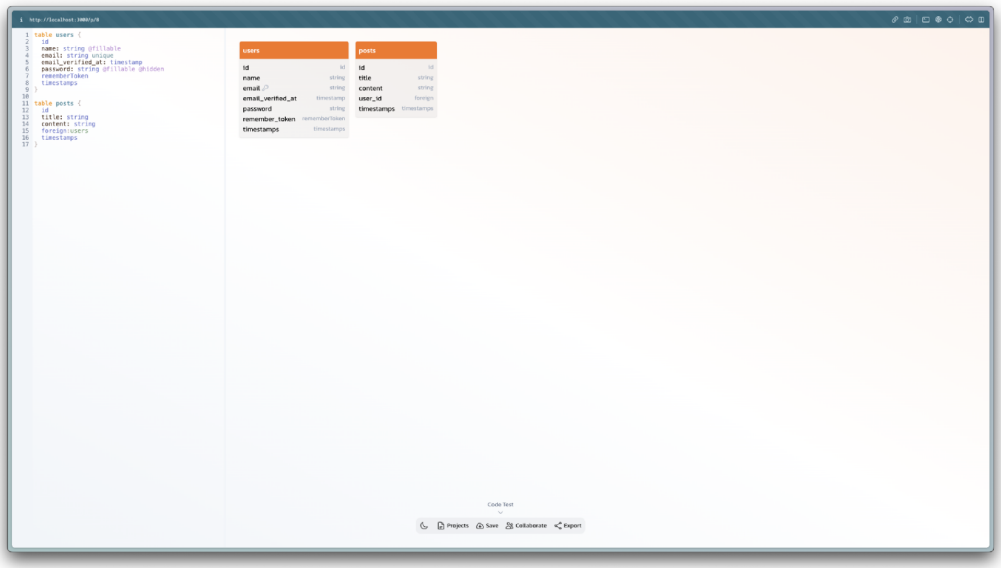


Obr. 4.4: Úvodná stránka so zoznamom projektov v tmavom režime

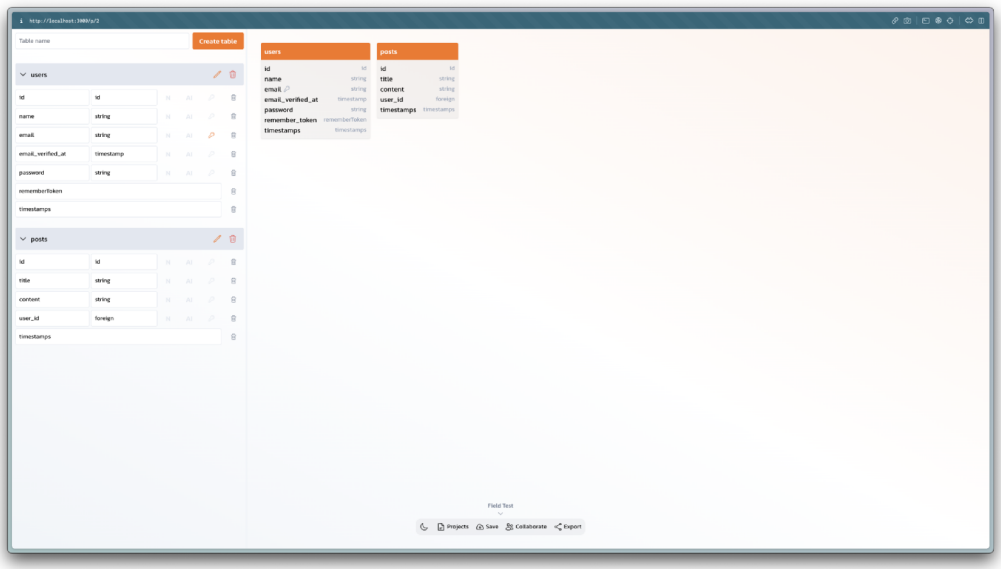
Ako posledná dôležitá stránka je editor, v ktorom sa vytvárajú konečné databázové schémy. Editor obsahuje po ľavej strane textové pole pre pseudo-kód alebo vstupné polia, podľa toho, ako si daný užívateľ zvolí. Po pravej strane sa nachádza náhľad jednotlivých tabuliek, ktoré boli vygenerované vďaka spomenutému pseudo-kódu alebo vyplnených vstupov.

Textové pole je upravené pomocou JavaScriptu tak, aby sa správalo prirodzenejšie pre programátorov. Má vstavané základné funkcie ako napríklad označenie riadkov, zvýrazne-

nie syntaxu, automatické odseky alebo doplňovanie zátvoriek a úvodzoviek pre vyznačenie reťazca. Zvýraznenie syntaxu je sprostredkované vďaka balíčku pre JavaScript s názvom Prism.js[1]. Tento balíček obsahuje zvýraznenie syntaxe pre množstvo už existujúcich programovacích jazykov. Podporuje jazyky ako napríklad jazyk C, Python, C#, PHP a mnoho ďalších. Umožňuje taktiež konfiguráciu vlastného jazyka pomocou regulárnych výrazov, ktoré boli použité pri definovaní vlastného jazyka na tvorenie databázových schém.



Obr. 4.5: Vytváranie databázových schém v pseudo-kóde



Obr. 4.6: Vytváranie databázových schém pomocou vstupných polí

[1] https://prismjs.com/

4.2 Vytváranie databázových schém

Základom vytvárania databázových schém sú stĺpce v databáze, ktoré určujú ako a aké dáta sa budú do jednotlivých tabuliek ukladať. Pre nastavenie typu dát resp. stĺpcov, sa používajú konvencie podľa aplikačného rámca Laravel. Laravel podporuje väčšinu typov pre MySQL databázu. Taktiež obsahuje vstavané funkcie pre niektoré typy ako napríklad id, časy vytvorenia alebo upravenia záznamu. Typ id vygeneruje stĺpec s názvom id a dátovym typom BIG INTEGER, ktorému sa automaticky priradí primárny kľúč s atribútom AUTO INCREMENT, ktorý znamená, že sa bude zvyšovať hodnota stĺpca po každom pridaní záznamu. Tento stĺpec, id, je základom takmer všetkých tabuliek, ktoré sú vytvorené, preto webová aplikácia generuje tieto stĺpce automaticky bez nutnosti definovania tohto stĺpca. Ďalšie stĺpce, ktoré sa veľmi často využívajú sú spomenuté časy vytvorenia a upravenia záznamu. Vstavaná funkcia TIMESTAMPS vytvorí dva stĺpce, ktoré budú mať typ TIMESTAMP, ktoré určujú kedy bol záznam vytvorený a upravený. Typ TIMESTAMP je časová značka, ktorá určuje čas v sekundách od začatia unixového času[2] t.j. 1. Január 1970.

Po určení týchto dvoch bodov je ďalej možné využívať rôzne modifikátory alebo kľúče stĺpcov. Medzi modifikátory patrí napríklad predvolená hodnota stĺpca, možnosť nulovej hodnoty, nastavenie automatickej inkrementácie a jej prvotnú hodnotu alebo upravenie použitej znakovej sady.

Kľúče alebo indexy sú využívané veľmi často. Používajú sa hlavne kvôli optimalizáciám a zabezpečenie integrity dát. Indexy pomáhajú pri zrýchlení vyhľadávania záznamov v databáze, tým, že sa indexované stĺpce skopírujú do dátovej štruktúry, ktorou pri MySQL je to najčastejšie B-Strom, a jednotlivé hodnoty sa vyhľadávajú v danej štruktúre a neprechádza sa riadok po riadku v tabuľke. Indexy sa väčšinou používajú pre stĺpce, podľa ktorých identifikujeme daný záznam v tabuľke, resp. podľa ktorého chceme daný záznam vyhľadať. Primárne kľúče jednoznačne identifikujú každý riadok v tabuľke. Zabezpečuje to, že každý záznam v tabuľke má priradený svoj vlastný identifikátor a môžu byť použité pre vytvorenie vzťahu s inými tabuľkami. Unikátny kľúč je podobný primárnemu kľúču, avšak tento kľúč môže byť definovaný pre viaceré stĺpce, oproti primárnemu. Spatial index je typ indexu, ktorý využíva dotazy na základe polohy. Tieto indexy sú používané hlavne pri geografických informačných systémoch, ktoré vyhľadávajú dáta na základe polohy. Ďalším podporovaným indexom je FULLTEXT index. Vďaka FULLTEXT indexu je možné rýchlejšie nájsť záznam, ktorý obsahuje vyhľadávanú frázu v jednotlivých stĺpcoch aj v prípadoch, že daná fráza sa nachádza v strede uloženého textu v stĺpci.

Pseudo-kód

Pseudo-kód pre praktickú časť tejto bakalárskej práce bol navrhnutý tak, aby bol čo najjednoduchší na písanie. Syntax bol z časti inšpirovaný praktikami, ktoré využíva Laravel. v pseudo-kóde sa zapisujú jednotlivé tabuľky, v ktorých sú potom definované jednotlivé stĺpce. Tieto stĺpce obsahujú vo väčšine prípadoch názov stĺpca, po ktorom nasleduje typ stĺpca a prípadne modifikácie a indexy alebo kľúče. Nakoľko pomocou aplikácie je možné vytvoriť aj model pre Laravel, pridané sú taktiež pravidlá, nazývané at-rules.

Jednotlivé tabuľky sa vytvárajú pomocou kľúčového slova `table`, za ktorým sa musí , hneď po medzere, nachádzať názov. Názvy tabuliek sú odporúčané podľa konvencií Laravelu, použitím plurálu a v prípade viac slových názvov použiť oddeľovač _. Za definovaným

[2] https://websiteseochecker.com/blog/what-is-timestamp/

názvom tabuľky musí nasledovať minimálne jedná medzera a otvorená zložená zátvorka {. Následným novým riadkom a zatvorenou zloženou zátvorkou } sa tabuľka vytvorí.

```
1  table users {
2
3  }
```

Obr. 4.7: Ukážka definície tabuľky pomocou pseudo-kódu.

Po vytvorení tabuľky nasleduje definícia stĺpcov. Tieto stĺpce musia byť definované medzi otvorenou zloženou zátvorkou { a uzavretou zloženou zátvorkou }. Každý stĺpec je oddelený znakom nového riadku. Stĺpec môže začínať buď názvom stĺpca, po ktorom nasleduje znak :, alebo typ stĺpca. Niektoré typy ako napríklad `id` alebo `ipAddress` nepotrebujú definovať názov stĺpca, nakoľko Laravel automaticky tieto stĺpce definuje názvom `id` alebo `ip_address`. Existujú aj typy, pri ktorých názov stĺpca nemôže byť zadaný, nakoľko tieto typy vytvárajú viacero stĺpcov, alebo vlastný názov Laravel nepodporuje. Medzi tieto typy patria `rememberToken` a `timestamps` alebo `timestampsTz`. Typ `rememeberToken` vytvorí v databáze nulovateľný stĺpec s názvom `remember_token` a typom `string` a dĺžkou 100, ktorý slúži pre uchovávanie tokenov, pre užívateľov. Pre typ `timestamps` alebo `timestampsTz` sa vytvorí dvojica stĺpcov `created_at` a `updated_at`, ktoré majú typ `TIMESTAMP` a udávajú kedy bol záznam vytvorený a kedy upravený.

```
1  table users {
2    id
3    name: string @fillable
4    email: string unique @fillable
5    email_verified_at: timestamp nullable
6    password: string @hidden @fillable
7    rememberToken @hidden
8    timestamps
9  }
```

Obr. 4.8: Ukážka definície tabuľky so stĺpcami a s použitím modifikátorov, indexov a at-rule pravidiel v pseudo-kóde.

V prípade, že stĺpec má zadaný názov, na druhom mieste sa musí nachádzať typ stĺpca. Existuje množstvo typov stĺpca, ktoré sú podporované v MySQL databáze. Možné je zapísať napríklad stĺpce `string`, ktorý v MySQL vytvorí stĺpec s typom `VARCHAR`, `boolean`, známy taktiež ako `TINYINT` alebo typ `json`.

Ak je názov aj typ definovaný, nasledujú voliteľné modifikátory, indexy a kľúče, a at-rule pravidlá. Poradie týchto parametrov nezáleží, môžu byť definované v akomkoľvek poradí. Medzi identifikátory patrí napríklad `nullable`, `charset` alebo `from`, ktorý definuje začiatočné číslo pri automatickej inkrementácií. Medzi indexy patria `primary`, `index` alebo `unique`.

At-rule pravidlá sa zapisujú pomocou znaku @, kde za ním sa môže nachádzať hodnota `fillable` alebo `hidden`. Tieto pravidlá poskytujú aj v budúcich verziách väčšie možnosti

pri konfigurácií modelu. Pravidlo `fillable` vytvorí v modeli atribút `fillable`, do ktorého zapíše názov stĺpca. Tento atribút označuje, že je umožnené pre tento stĺpec hromadné prideľovanie, čo v Laraveli znamená, že je možné do tohto stĺpca zapisovať hodnoty. Laravel využíva tieto atribúty hlavne kvôli zvýšenej bezpečnosti, pričom znemožňuje upravovať stĺpce, ktoré nie sú hromadne prideľovateľné. Druhá hodnota `hidden` zabezpečí, že model obdrží atribút `hidden` s hodnotou názvu stĺpca, čo znamená, že hodnota sa bude vynechávať z vyžiadaných dotazov z databáze. Stĺpce s týmto atribútom sa využívajú hlavne pri heslách, kedy nechceme priamo na frontend posielať s informáciami o užívateľoch aj ich heslá.

Po vytvorení tabuľky, je možné vytvárať ďalšie rovnakým princípom. Pričom jednotlivé tabuľky musia byť oddelené aspoň jedným znakom nového riadku.

Vstupné polia

Vytváranie databázových schém je umožnené taktiež pomocou vstupných polí. Princíp je podobný, ako pri využití pseudo-kódu, s rozdielom, že všetky tabuľky a stĺpce sú reprezentované vstupmi, ktoré sa jednoducho vyplnia, prípadne kliknú na niektorú z vlastností. Tabuľky je možné vytvárať, mazať a upravovať. Stĺpce, ktoré patria v tabuľke je možné pomenovávať, nastavovať typ, a na výber je taktiež pár modifikácií. Oproti kódu sú modifikácie obmedzené a zjednodušené, pre lepšiu orientáciu v užívateľskom rozhraní. Dôležité a často používané modifikácie ako napríklad nulový stĺpec, automatická inkrementácia alebo kľúče a indexy, sú dostupné aj vo vstupných poliach. Obmedzenie sa týka hlavne modifikácií ako nastavenie znakovej sady, postavenie stĺpca v tabuľke alebo nastavenie začiatočnej hodnoty inkrementácie.

Po zadaní názvu stĺpca je možné vybrať si typ dát, ktoré bude uchovávať. Tento zoznam obsahuje aj funkcie spomenuté pri pseudo-kóde, ktorými sú `timestamps`, `timestampsTz` a `rememberToken`. Typ stĺpca `id` je taktiež zahrnutý, pri ktorom je automaticky vyznačená automatická inkrementácia. Niektoré typy stĺpcov, ako pri pseudo-kóde, majú automaticky dopĺňané názvy. Medzi tieto patrí napríklad typ `ipAddress` alebo `macAddress`.

Vstupné polia podporujú všetky typy kľúčov a indexov, ktoré sú napríklad `primary`, `unique` alebo `index`. Nastavenie vzťahu medzi tabuľkami, teda použitie `foreign`, je možné pomocou definovania typu stĺpca ako `foreign` a následne zo zoznamu vybrať tabuľku, pre ktorú sa má vytvoriť vzťah. Názov stĺpca môže a nemusí byť určený nakoľko sa, v prípade nevyplnenia, generuje automaticky. Ak tento vzťah vyžaduje aj cudzí kľúč, je možné vybrať v zozname kľúčov cudzí kľúč, teda `Foreign key`, čím sa v databázy, po migrácií, vytvorí cudzí kľúč s referenciou na danú tabuľku. Medzi obmedzenia patrí taktiež nemožnosť definovať udalosti po vymazaní alebo upravení referenčnej tabuľky.

4.3 Export

V prípade, že databázová schéma je pripravená pre použitie, pomocou exportu získame všetky potrebné modely a migrácie. Stiahne sa súbor s príponou `.zip`, ktorý obsahuje dané súbory v hierarchickej štruktúre, akú používa práve Laravel. Tieto súbory je napokon možné presunúť do už existujúceho projektu. Pri exporte pseudo-kódu sa vždy použije posledný úspešne skompilovaný kód. Všetky chyby, ktoré sú aktuálne v kóde nebudú vo výsledných súboroch.

4.4 Kooperácia

Kooperácia v aplikácií zohráva dôležitý faktor, aby mohli užívatelia spolupracovať na jednom projekte. Umožnená je vďaka WebSocketom, ktoré sú na frontende spojazdnené pomocou spomínaných knižníc socket.io a Laravel Echo. Užívateľ si môže zvoliť, s kým bude kooperovať. Hostí alebo kooperátorov si užívateľ pozýva na základe mailu. Pozývať je možné v zozname projektov na hlavnej stránke alebo priamo v editore pomocou plávajúceho menu. v menu, ktoré sa otvorí sa zobrazia všetci kooperátori, ktorí majú prístup k danému projektu, teda databázovej schéme. v prípade, že menu je otvorené na stránke editora, je taktiež vidieť, koľko ľudí aktuálne sa nachádza v editore.

Pomocou oprávnení, pri pozvaní alebo úprave kooperátorov, možno vybrať dva typy oprávnení. Prvým oprávnením je zobrazenie, ktoré povolí druhému užívateľovi zobrazovať databázovú schému, ale nebude môcť s ňou manipulovať a ani upravovať projekt. Druhé oprávnenie je úprava, pomocou ktorej pozvaný užívateľ dostane práva k úprave databázovej schémy. v prípade uloženia, vďaka WebSocketom, ostatným užívateľom príde notifikácia, že projekt bol aktualizovaný a automaticky sa im upraví kód, po čom sa vygenerujú nové tabuľky v schéme. Pomocou notifikácií sú taktiež ohlásené odchody a príchody jednotlivých užívateľov.

WebSockety využívajú jeden kanál a dve udalosti, `update` a `join`. Udalosť `update` sa zachytáva vďaka skrytému kanálu, ktorý vyžaduje autorizáciu, čo znamená, že užívateľ musí byť prihlásený. Táto udalosť zachytáva úpravy a ich následné uloženie v editore. Na podobnom princípe funguje aj druhá udalosť `join`. Táto udalosť však využíva kanál prítomnosti, ktorý zaznamenáva jednotlivé príchody a odchody. Pre zistenie, kto je aktuálne v miestnosti sa využíva databáza Redis.

Kapitola 5

Implementácia riešenia

Táto kapitola obsahuje postup a implementáciu navrhnutého riešenia. Preberaná je jednotlivá štruktúra, klientska časť, teda frontend a serverová časť, backend. Zaoberá sa taktiež popisom kompilátora, ktorý bol použitý pre preklad pseudo-kódu do jednotlivých databázových súborov.

5.1 Štruktúra

Aplikácia sa skladá z dvoch častí, ktorými sú frontend a backend. Každá časť zohráva svoju dôležitú rolu. Frontend bol písaný v Reacte, ktorý sa kompiluje pomocou lokálneho vývojového nástroja Vite[1]. Backend som písal v spomínanom aplikačnom rámci Laravel.

Štruktúra frontendu

Frontend je štruktúrovaný na základe dlho zaužívaných pravidiel a konvencií pri vytváraní aplikácií v knižnici React.js. Obsahuje dve hlavné zložky, ktorými sú `public` a `src`. v `public` zložke môžu byť uchované obrázky aplikácie alebo iné pomocné súbory. v danej aplikácií táto zložka nie je moc využívaná, nakoľko aplikácia neobsahuje žiadne obrázky ani nutnosť využívať iné súbory. Zložka `src` obsahuje komponenty, jednotlivé stránky, pomocné funkcie, globálne stavy alebo router, ktorý sa využíva pri zobrazovaní obsahu na konkrétnych cestách URL. Počiatočný bod sa nachádza v `src/main.tsx`, v ktorom je inicializovaná aplikácia, recoil a router. Recoil používam práve pre správu globálnych stavov, ktoré budú spomenuté neskôr v tejto kapitole. Router je nakonfigurovaný tak, aby obsluhoval všetky nakonfigurované cesty, teda stránky. Jednotlivé stránky sa nachádzajú v zložke `src/pages`, ktoré využívajú jednotlivé komponenty zo zložky `src/components` a prípadne volanie API zo zložky `src/api`. Aplikáciu som rozdelil do troch hlavných rozložení, ktorými sú aplikačné, prihlasovacie a rozloženie editora. Každé toto rozloženie sa používa pri určitých stránkach, aby bolo pre danú sekciu stránky jednotné.

Pre maximálnu voľnosť pri vytváraní komponentoch využívam CSS rámec tailwindcss[2]. Tailwindcss je utility-first rámec, pre kaskádové štýly, ktorý obsahuje nízkoúrovňové triedy, vďaka ktorým umožní vysokú flexibilitu a rýchlosť pri vývojí užívateľských rozohraní. Neobsahuje celistvé komponenty ako napríklad bootstrap[3], ktorý je pri vývojí síce rýchlejší, ale nemá takú voľnosť pri dizajnovaní ako práve tailwindcss.

[1] https://vitejs.dev/
[2] https://tailwindcss.com/
[3] https://getbootstrap.com/

Základnú štruktúru frontendu v Reacte som si vybral teda nasledovnú:

```
src
├── api ..... Zložka obsahujúca všetky funkcie pre komunikáciu s backendom
├── components ..... Všetky komponenty aplikácie
├── hooks ..... Vlastné definované React hook-y
├── layouts ..... Definícia rôznych rozložení
├── pages ..... Jednotlivé stránky aplikácie
├── recoil ..... Globálne stavy
├── styles ..... Hlavné štýly aplikácie, kde je inicializovaný aj tailwindcss
├── types ..... Globálne definované typy
└── utils ..... Pomocné funkcie využívané v aplikácií.
```

Pre dôkladnejšie programovanie a zabezpečenie správnych typov používam `TypeScript`. Tento balíček je veľmi populárny tým, že prináša typovanie pre javascript. Zabezpečuje typovú korektnosť a kontroluje súbory, aby mala každá premenná a funkcia správne typy, či už v priradení premennej alebo premenná ako parameter funkcie. React používa spomínané `JSX`, vďaka ktorému dokáže vkladať HTML elementy priamo do javascriptu, avšak pre `typescript` sa používa `TSX`, čo reprezentuje TypeScript XML.

Štruktúra backendu

Štruktúra backendu je robustnejšia ako pri frontende. Základná štruktúra po nainštalovaní Laravel aplikácie je nasledovná:

```
├── app ..... Zložka obsahujúca hlavné časti aplikácie
│   ├── Console ..... Definícia vlastných príkazov pre Artisan
│   ├── Exceptions ..... Aplikačné výnimky
│   ├── Http .. Definícia kontrollerov, midlvérov, validácia požiadavkou a jadro aplikácie
│   ├── Models ..... Definované modely pre databázu
│   └── Providers ..... Poskytovatelia služieb, všetky registrované komponenty
├── bootstrap ..... Spúšťač HTTP jadra a konzolového jadra
├── config ..... Hlavná konfigurácia aplikácie.
├── database ..... Potrebné súbory pre správu databáz
│   ├── factories ..... Továrne pre jednotlivé modely
│   ├── migrations ..... Migračné súbory
│   └── seeders ..... Naplňovače databáz s testovacími dátami
├── public ..... Zložka prístupna pre užívateľa, z ktorej sa spúšťa celá aplikácia
├── resources ..... Potrebné súbory pre vývoj frontendu
│   ├── css ..... Súbory kaskádových štýlov
│   ├── js ..... JavaScript súbory
│   └── views ..... Views aplikácie Laravel
├── routes ..... Definované koncové body, ktoré má aplikácia registrované
├── storage ..... Úložisko pre aplikáciu
├── tests ..... Tvorenie testov pre aplikáciu
├── .env.example ..... Vzorový enviromentálny súbor, pre konfiguráciu prostredia
└── artisan ..... Jadro nástroja príkazového riadku
```

Hlavné zložky, ktoré som používal sú `app`, `database` a `routes`. v prípade, že frontend by bol vytvorený v rámci Laravelu, využíva sa taktiež zložka `resources`, ktorá obsahuje všetky potrebné súbory pre frontend ako napríklad blade šablóny, kaskádové štýly a skripty písané v javascripte.

Do zložky `app` som ukladal všetky potrebné súbory pre spracovanie požiadavkou. v tejto zložke využívam hlavne pod zložku `Http`, do ktorej ukladám všetky kontroléry. Vytvoril som si taktiež zložku `Enums`, do ktorej ukladám enumerácie, ktoré sú dostupné od PHP verzie 8. Podzložka `Events` obsahuje vysielacie udalosti, ktoré pracujú s WebSocket-mi. Existujú tri súbory s udalosťami `EditorCodeUpdated.php`, `EditorFieldsUpdated.php` a `EditorUserJoined.php`. Pre autorizáciu prichádzajúcich požiadavkou používam auzorizačnú politiku poskytovanú Laravelom. Jednotlivé politiky obsahujú sedem základných autorizačných typov pre kontrolér typu `API Resource`[4]. Vytvoril som si taktiež podzložku `Services`, ktorá obsahuje zdrojové kódy kompilátora, generátora tabuliek pomocou vstupných polí a správcu povolení. Podzložka `Traits` obsahuje pomocné funkcie pre odpovede smerované ku klientovi z jednotlivých kontrolérov.

Definícia všetkých koncových bodov, na ktoré užívateľ pristupuje sú vypísané v súbore `routes/api.php`. Tieto koncové body sú verzované a s prefixom ako `api/v1`. Pre autentifikáciu sa používa balíček `Laravel Sanctum`, ktorý zabezpečuje prihlasovanie a odhlasovanie užívateľa. Databáza a jej jednotlivé tabuľky a stĺpce, sú nakonfigurované v zložke `database`, kde sú nastavené jednotlivé migrácie, továrne a seedery, vďaka ktorým sa naplňujú testovacie dáta do databázy, ktoré boli využité pri testovaní.

Pre správne fungovanie WebSocketov v aplikácií, bolo nutné využiť balíček `predis`, ktorý sa následne nastavuje v konfigurácií redis klienta. v Laraveli je konfigurácia rôznych súčastí aplikačného rámca možná s pomocou konfiguračných súborov v zložke `config` a prostredníctvom enviromentálneho súboru `.env`.

5.2 Klientska časť

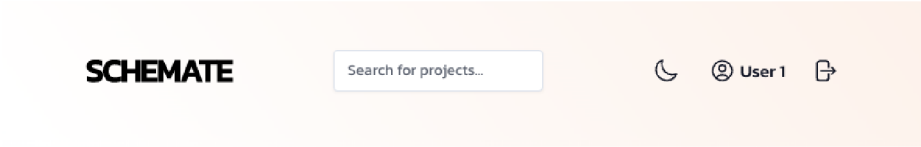
Klientska časť začína pri prihlásení alebo registrácií užívateľa. Po úspešnom prihlásení následne užívateľ prichádza do dashboardu, kde uvidí prípadný zoznam projektov, ak má už vytvorený projekt, alebo môže vytvoriť nový.

Po vytvorení tohto projektu je užívateľ presmerovaný k editoru, ktorý zobrazí obsah na základe typu projektu. v prípade typu projektu `Code` bude zobrazený editor kódov, ktorý slúži na vytváranie databázových schém pomocou definovaného pseudo-kódu alebo typ projektu `Fields`, pre vytváranie prostredníctvom vstupných polí. Vytvorený diagram, ktorý je pripravený pre použitie je následne možné exportovať.

Dashboard

Dashboard je počiatočným bodom, k tvoreniu databázových schém. Na dashboarde sa nachádza navigačná lišta obsahujúca vyhľadávacie pole, prepnutie témy z klasickej na tmavú a naopak, úprava užívateľa, a tlačidlo odhlásenia.

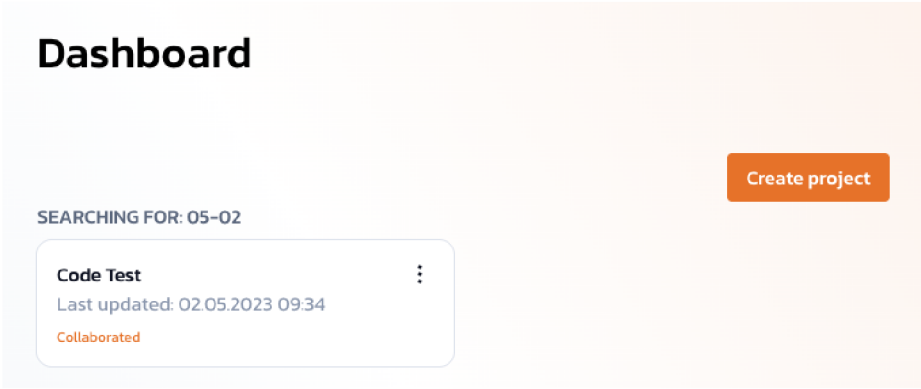
[4] https://laravel.com/docs/10.x/controllers#api-resource-routes



Obr. 5.1: Navigačná lišta v dashboarde

Vyhľadávaie projektu

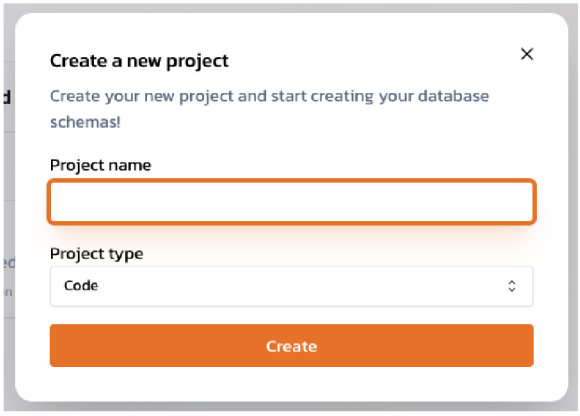
Projekt sa vyhľadáva prostredníctvom vyhľadávacieho poľa, do ktorého sa zadajú iniciálky názvu projektu, alebo dátum vo formáte `YYYY-MM-DD HH:MM`. Po chvíli od zadaní iniciálok sa v dashboarde zobrazia iba projekty, ktoré zodpovedajú vyhľadávaniu. Vyhľadávane je oneskorené, kvôli tomu, že nie je ideálne po každom znaku posielať požiadavok na server. Používa sa balíček `lodash` a konkrétne funkcia `debounce`, ktorá zavolá API, iba v prípade, že do vstupného poľa nebol zadaný žiadny znak po zvolenú dobu.



Obr. 5.2: Vyhľadávanie podľa dátumu poslednej úpravy projektu

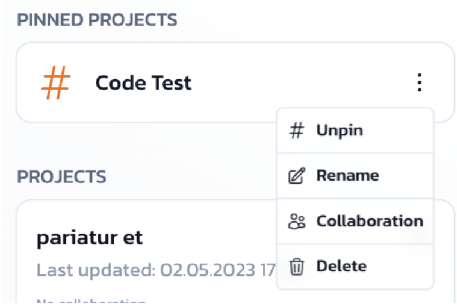
Vytvorenie a správa projektu

Projekt sa vytvára pomocou tlačidla `Create project`, po ktorom sa užívateľovi zobrazí dialóg, kde môže zadať názov a typ projektu. Typ projektu má dve možnosti a to `Code`, vytváranie databázových schém pomocou pseudo-kódu a `Fields` pomocou vstupných polí. Po výbere typu projektu už nie je možné zmeniť tento typ.



Obr. 5.3: Dialóg vytvárania nového projektu

Spravovanie projektov je umožnené prostredníctvom symbolu troch bodiek pri projekte. Po kliknutí sa zobrazí menu, v ktorom sa môže pripnúť projekt na vrch, premenovať projekt, spravovať kolaboráciu alebo zmazať projekt.

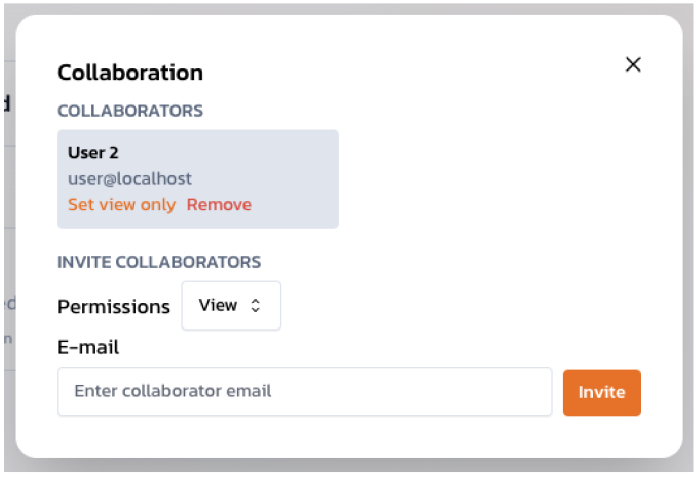


Obr. 5.4: Spravovanie projektu pomocou rozbaľovacieho zoznamu

Spravovanie kooperácie

Kooperácie je možné vytvárať pomocou známeho emailu druhého používateľa, ktorého chceme pozvať do kolaborácie. Pri vytváraní pozvánky sa zadávajú aj povolenia, ktorými určujeme, či užívateľ bude môcť iba zobrazovať projekt alebo aj upravovať. Následne po vytvorení pozvánky užívateľ sa zaradí medzi kolaborantov, kde bude vidno jeho meno, email a možnosti prepínania práv prípadne odstránenie z kolaborácie.

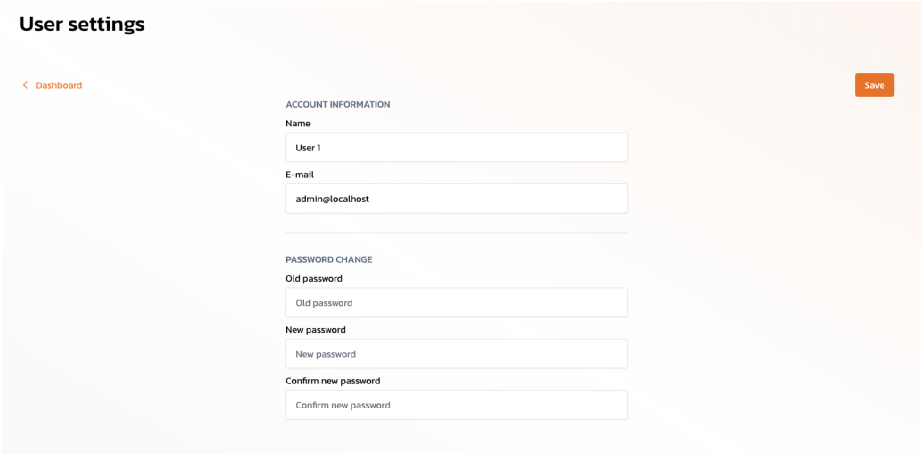
Pre rozlíšenie jednotlivých projektov, ktoré sú a nie sú v kooperácií, je pod posledným dátumom úpravy prípadný nápis od koho je projekt zdieľaný. Pri zdieľanom projekte od iného užívateľa je tento projekt po nabehnutí myšou vyznačený zelenou farbou.



Obr. 5.5: Dialóg spravovania kooperácií

Úprava užívateľa

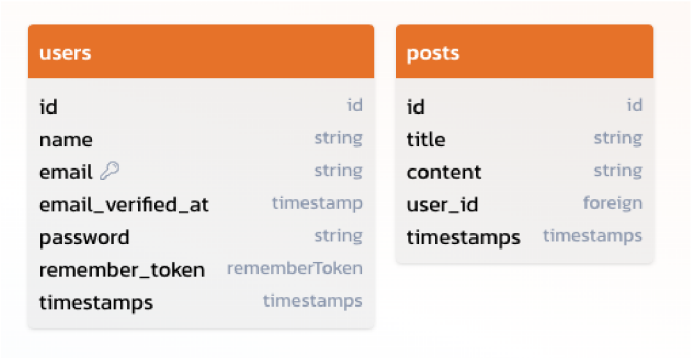
Aplikácia poskytuje aj úpravu užívateľských údajov, medzi ktoré patrí meno, email a heslo.



Obr. 5.6: Úprava užívateľa

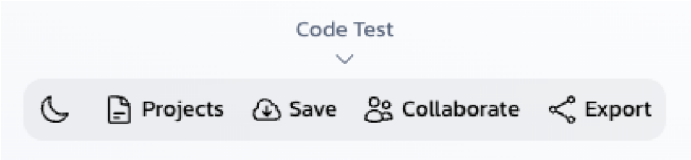
Editor

Samotný editor databázovej schémy obsahuje dva komponenty. Prvý komponent je editor kódu alebo vstupné polia, záleží od typu projektu. Druhým komponentom je generátor diagramov, ktoré sa generujú automaticky ihneď po uložení projektu. Diagramy sú vygenerované na backende, kde sa kód alebo vstupné polia spracujú a odošle sa odpoveď s už vytvorenými tabuľkami. Tabuľky obsahujú názov, jednotlivé stĺpce a ich typy. v prípade, že stĺpec má priradenú modifikáciu alebo kľúč, zobrazí sa ikonka vedľa názvu stĺpca.



Obr. 5.7: Vygenerované tabuľky pomocou pseudo-kódu

Všetky potrebné funkcie a nastavenia ako vrátenie na dashboard, ukladanie projektu, spravovanie kooperácií a export sú dostupné pomocou plávajúceho panela, ktorý sa nachádza na spodnej časti editora.



Obr. 5.8: Multifunkčný plávajúci panel v editore.

Možnosti vzdialenej kooperácie som spravil prostredníctvom WebSocketov s použitím balíčkov `socket.io-client` a `laravel-echo`. Balíček `laravel-echo` manažuje celé pripojenie a načúvanie kanálov, na ktoré backend odosiela správy a `socket.io-client` používa ako sprostredkovateľa pripojenia. Pri ukladaní sa používa klasický typ kanála, ktorý pri udalosti upraví databázovú schému všetkým pripojeným užívateľom. Pri zistení počtu pripojených užívateľov sa používa tzv. kanál prítomností. Tento kanál obsahuje tri metódy `here`, `joining` a `leaving`. Metóda `here` sa volá iba klientovi, ktorý sa aktuálne pripojí. Vďaka nej zisťujem, kto všetko sa aktuálne nachádza v miestnosti. v tú istú chvíľu sa ostatným pripojeným užívateľom volá metóda `joining`, ktorá oboznámi už napojených užívateľov, kto nový sa pripojil. Naopak pri odchode užívateľa z kanálu, sa všetkým, okrem odpojeného užívateľa, zavolá metóda `leaving`, ktorou zase oboznámi o odpojení.

Popis regulárnych výrazov pri pseudo-kóde

Editor kódu obsluhuje už spomínaný `prismjs`, ktorý vyznačuje syntax. Pre tvorenie vlastného jazyka, boli použité regulárne výrazy, vďaka ktorým sa farba líši pri jednotlivých príkazoch.

Definícia tabuľky

Začiatok definície tabuľky musí vždy obsahovať kľúčové slovo `table`, po ktorom nasleduje medzera a ľubovoľný názov tabuľky obsahujúci malé a veĺké písmená bez diakritiky. Pre oddelenie jednotlivých slov sa používa podčiarkovník `_`. Regulárny výraz teda vyzerá takto:

`/table [a-zA-Z_]+/` (`table users`, `table posts`...)

Názov stĺpca

Názov stĺpca sa identifikuje pomocou dvojbodky na konci sekvencie písmen a podčiarkovníku. Stĺpec môže byť pomenovaný malými alebo veľkými písmenami oddelené prípadným podčiarkovníkom, ktorý je použitý namiesto medzeri. Definícia stĺpca podľa konvencií Laravelu je zvyčajne malé písmo oddelené podčiarkovníkom ako napríklad `created_at`, ale umožnený je aj zápis v štýle `createdAt`. Regulárny výraz názvu stĺpca je popísaný nižšie:

`/[a-z_]+:/i` (`email:`, `user_id:`...)

Typ stĺpca

Typ stĺpca má vyššiu prioritu ako názov stĺpca, nakoľko taktiež môže obsahovať `:`, kvôli použitiu parametrov. Nakoľko typ stĺpca musí byť definovaný až za názvom stĺpca, prípadne na začiatku riadku, pri regulárnom výraze sa využíva negatívny pohľad dozadu použitím `(?<!...)`, kde definícia toho čo sa nesmie pred výrazom nachádzať sa určuje v mieste `...`. Pre kontrolu medzery alebo iného bieleho znaku sa používa token `\S`, ktorý určuje, že znak je hocijaký iný znak okrem bieleho. Nakoľko parametre typu, musia byť umiestnené neprostredne za znakom dvojbodky, používa sa token negatívneho pohľadu vpred `(?!...)`. v tomto prípade vďaka pohľadu vpred sa určuje, či typ obsahuje parametre a treba ich syntakticky vyznačiť, alebo obsahuje medzeru alebo iný biely znak a nasledujúce znaky nepatria pod parametre typu. Definícia regulárneho výrazu je v tomto prípade zložitejšia a je popísaná následovne:

`(?<!\S)(TYP1|TYP2|TYP3...)((?!\S)|:[a-zA-Z0-9,_]+)` (`string:255`, `time`...)

Kde `TYP1`, `TYP2`, `TYP3` označuje typ stĺpca, ktorý je definovaný v poli všetkých typov. Všetky podporované typy sú popísané v dokumentácii[5] Laravelu s výnimkou typov, ktoré nie sú definované pre databázy MySQL a typov `foreignUlid` a `foreignUuid`. Typ `foreignIdFor` je prístupný pomocou skráteného výrazu `foreign`.

Modifikátory, indexy a kľúče

Modifikátory a indexy sú popísané rovnakým spôsobom ako typy stĺpca. Podporované modifikátory, indexy a kľúče sú definované taktiež v Laravel dokumentácii[6].

Pravidlá at-rule

Pravidlá at-rule sú aktuálne používané pre vlastnosti modelu, ktoré určujú, či je daný stĺpec zapisovateľný a skrytý pri odosielaní na frontend. Tieto pravidlá majú určený všeobecný regulárny výraz, kde pravidlo musí začínať znakom `@` a ďalej môže obsahovať akýkoľvek znak mimo bielych znakov. Regulárny výraz pre pravidlá at-rule je popísaný následovne:

`/@\S*/` (`@fillable`, `@hidden`)

[5] https://laravel.com/docs/10.x/migrations#available-column-types
[6] https://laravel.com/docs/10.x/migrations#column-modifiers

Reťazce, čísla a interpunkcia

Reťazce sú v tejto aplikácií identifikované klasicky textom bez medzier a iných bielych znakov, alebo v prípade nutnosti použitia medzier sa môže použiť citovaný reťazec s použitím apostrofu `'`. Regulárny výraz pre definíciu reťazca a citovaného reťazca:

`/'[^']*'/` (`'string with space'`, `noSpaceString`)

Čísla sú podporované iba celé bez použitia desatinnej čiarky:

`/\d+/` (`123`)

Interpunkcia zahrňuje znaky `{` a `}`, ktoré ohraničujú definovanie tabuľky:

`/[{}]/`

5.3 Serverová časť

Serverová časť obsluhuje všetky prichádzajúce požiadavky od klienta. Každá požiadavka je obsluhovaná konkrétnym kontrolérom a jeho metódou. Pre každú cestu, okrem prihlásenia a registrácie, je priradený midlvér `auth:sanctum`, ktorý autorizuje užívateľa.

Autorizácia užívateľa

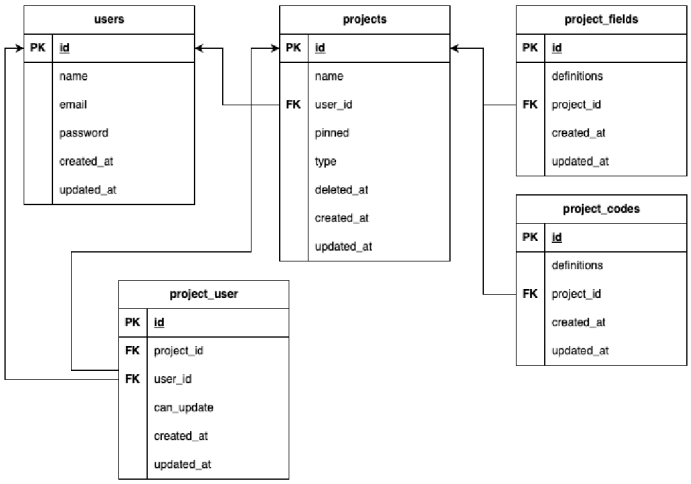
Autorizáciu používateľa som spravil na backende pomocou PHP relácií, ktoré sa vždy vytvoria po prihlásení užívateľa. Vďaka týmto reláciam mi umožňuje využiť aj Laravel ochranu proti spomínanému `CSRF`, nakoľko požiadavky obsahujú aj hlavičku s `CSRF` tokenom, ktorý sa validuje. Autorizácia prebieha pri každom požiadavku a taktiež v priebehu používania webovej aplikácie pomocou frontendového balíčka `swr`[7]. Pre autentizáciu užívateľa je vytvorená nasledujúca cesta:

- `GET` /user

Štruktúra databázy

Databáza obsahuje dokopy päť tabuliek. Jedná sa o tabuľky uchovávajúce užívateľov, jednotlivé projekty a pomocné tabuľky pre typy projektov a v neposlednom rade kontingenčnú tabuľku `project_user` medzi užívateľom a projektom, ktorá slúži pre zaznamenávanie kolaborácií.

[7] https://swr.vercel.app/



Obr. 5.9: Štruktúra databázy

Tvorba a správa projektov

Keď sa pri vytváraní projektu vyberá typ, zisťuje sa tento typ na backende a podľa toho sa vytvorí potrebný záznam v databáze. Pre typ projektu sa používajú tabuľky `project_code` a `project_fields`, ktoré sú prepojené s tabuľkou `projects` pomocou cudzieho kľúča. Po vytvorení sa vytvorený projekt odošle na frontend, aby sa mohol inicializovať. Kontrolér `ProjectController` obsluhuje nižšie popísané cesty:

- `GET` /projects - vypíše zoznam všetkych existujúcich projektov
- `POST` /projects - pridá nový projekt do databázy
- `GET` /projects/{project} - vypíše špecifický projekt
- `PUT` /projects/{project} - upraví špecifický projekt
- `DELETE` /projects/{project} - vymaže špecifický projekt

Pri editácií projektu sa o ukladanie stará jeden z dvojice kontrolérov, ktorými sú `ProjectCodeController` alebo `ProjectFieldsController`. Tieto kontroléry majú definované dve metódy `update` a `show`, ktoré používam na úpravy a zobrazenie vygenerovaných tabuliek. Cesty k týmto kontrolérom sú nasledovné:

- `GET` /projects/code/{project} - skompiluje a odošle kód a vytvorené tabuľky
- `PUT` /projects/code/{project} - skompiluje a uloží nový kód do databázy
- `GET` /projects/fields/{project} - vypíše uložené tabuľky z databázy
- `PUT` /projects/fields/{project} - vygeneruje a uloží tabuľky do databázy

Vytvorené databázové schémy sú spracované pri uložení na backende. Pseudo-kód navyše obsluhuje aj kompilátor, ktorý vytvára štruktúru tabuliek. Vstupné polia sú jednoducho spracované na základe užívateľských vstupoch, ktoré sú radené vo viacrozmerných poliach a následne sú spracované do jednotlivých tabuliek.

Tabuľky sú tvorené pomocou tried. Každá tabuľka vytvorí objekt z triedy `Table`, ktorá obsahuje názov, model a definíciu stĺpcov. Model tabuľky sa nikde nezadáva, pretože ten automaticky generujem na backende pomocou triedy `Pluralizer`, ktorú poskytuje Laravel. Táto trieda dokáže vytvárať jednotné aj množné číslo z anglických slovíčok a taktiež vytvárať rôzne typy zápisov ako napríklad tabuľový alebo "classify".

Stĺpce zahŕňajú názov, typ a jeho parametre, modifikácie, indexy, kľúče a atribúty pre model `fillable` a `hidden`. Súčasťou je taktiež definovanie cudzieho kľúča, ktorý má definovanú tabuľku a stĺpec, na ktorý má referenciu. Uchováva aj akcie `onUpdate` a `onDelete`, ktoré definujú, čo sa má stať so záznamom, v prípade vymazania alebo upravenia záznamu, ktorý je vo vzťahu.

Vytvorené tabuľky v tejto štruktúre sú následne odoslané na frontend, ktorý s nimi pracuje ďalej.

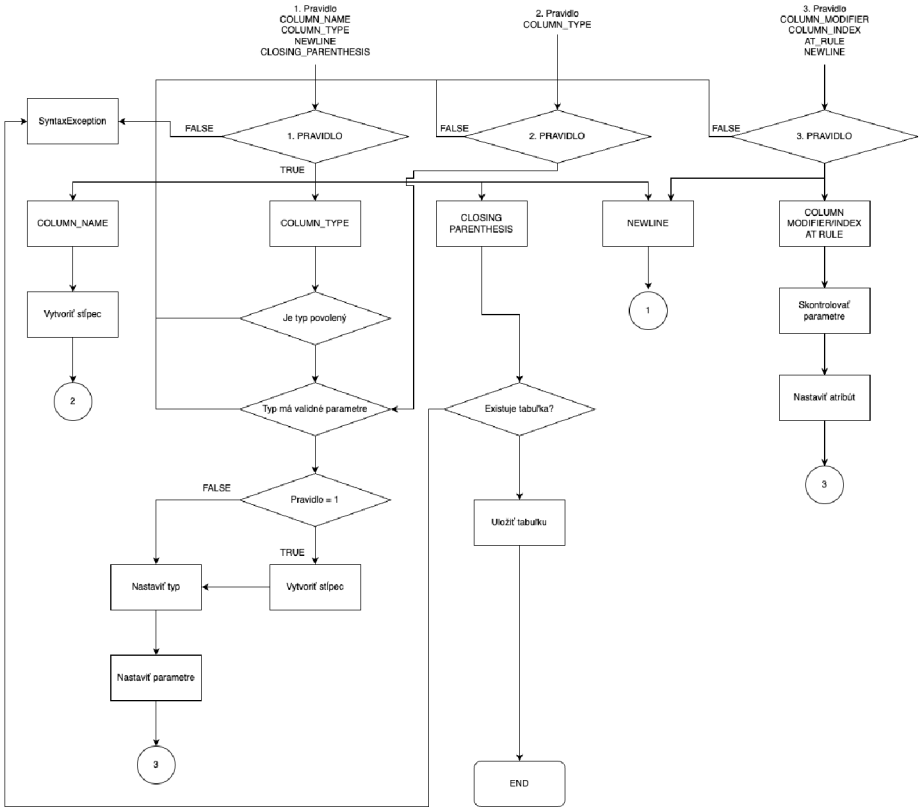
Kompilátor pseudo-kódu

Kompilátor disponuje troma metódami, pri kompilovaní kódu, ktorými sú `tokenizer`, `syntaxAnalysis` a `generate`. `Tokenizer` načíta a analyzuje vytvorený kód na frontende. Počas analýzy sa vytvárajú tokeny, ktoré sú oddelené medzerou, novým riadkom alebo tabulátorom. Typ tokenov vzniká na základe ich obsahu. Token môže byť napríklad kľúčové slovo `table`, typ stĺpca `string` alebo modifikátor `unique`. Analýza uchováva aj údaje, na akom riadku a stĺpci sa konkrétne token nachádza, aby som vedel pri frontende, na ktorom riadku sa nachádza chyba.

Po zozbieraní tokenov, zavolám metódu syntaktickej analýzy `syntaxAnalysis`. Syntaktická analýza má definované práve dve pravidlá, podľa ktorých sa riadí. Každé pravidlo má určené typy tokenov, ktoré v poradí akceptuje. Validácia týchto tokenov je definovaná v abstraktnej triede `RuleDefinition`. Po validácií tokenu sa volá metóda `validated`, ktorá ako parameter prijíma validovaný token. Túto metódu používam hneď po validácií tokenu, aby som vykonal potrebné akcie pre vytvorenie tabuľky a inicializovanie stĺpca.

Pravidlá pre definíciu tabuľky sú v súbore `TableRuleDefinition`. v tomto súbore po validácií pridávam do statického repozitára vytvorenú tabuľku. Tabuľka má určitý názov a model, ktorý sa generuje spomínaným Pluralizérom. Po uložení tabuľky do repozitára k nej pristupujem pomocou triedy a jej metódy `TableRepository::getTable()`. Túto metódu neskôr používam pri pridávaní jednotlivých stĺpcov do tabuľky.

Druhý súbor pravidiel `ColumnRuleDefinition` sa stará o to, aby po validácií tokenu sa pridali jednotlivé stĺpce do repozitára s tabuľkami a následne sa tabuľka uložila. Pravidlá pre jednotlivé stĺpce sú o niečo zložitejšie ako pri definícií tabuliek, preto som si vytvoril vývojový diagram v obrázku 5.10. Niektoré typy ako `id`, `ipAddres` alebo `softDeletes` majú automaticky zadané názvy stĺpcov, tým pádom ich nie je nutné zadávať.



Obr. 5.10: Vývojový diagram pre postupnosť tokenov v pravidle pre definíciu stĺpca

Generátor súborov

Generátor používa vytvorenú štruktúru tabuliek, ktorá je dostupná v repozitári. Pre každú tabuľku, zavolám metódy `makeMigration` a `makeModel`, s ktorými vytvorím obsah súborov a následne priradím do poľa so všetkými súbormi, ktoré sa majú vygenerovať. Pri generovaní používam predom vytvorené súbory, tzv. stubs. Do týchto súborov som si predpripravil šablóny migrácie a modelu, a pridal premenné zapísané medzi zloženými zátvorkami. Tieto premenné, napokon prepisujem s obsahom, ktorý vygeneroval generátor. Tento generátor, sa spúšťa pri zavolaní exportu databázovej schémy na cestu:

- `GET` /projects/{project}/export

Po zavolaní tejto cesty skompilujem kód a vygenerujem obsah so súbormi, pomocou generátora. Všetky súbory sa vytvoria v dočasnej ceste na backende a zazipujú. Celý súbor zip, potom pošlem na frontend, ktorý ho spracuje a následne dá sťahovať.

Kolaborácia

Kolaboráciu spravuje samostatná tabuľka `project_user`, ktorá vytvára vzťah medzi projektom a užívateľom, tzv. kontingenčná tabuľka. Pri vytváraní kolaborácie, získavam užívateľa prostredníctvom emailu a autorizujem požiadavku, či môže byť spracovaná. Pri kolaboráciách používam databázový systém Redis spolu so serverom Laravel Echo Server. Tento server beží na technológií Node.js[8], čo je prostredie pre vývoj webových aplikácií pomocou

[8] https://nodejs.org/en

JavaScriptu. Laravel Echo Server je nakonfigurovaný na port `6001`, na ktorom komunikuje s frontendom. Nasledne komunikuje so serverom Redis, bežiaci na porte `6379`, ktorý slúži pre neprerušené spojenie medzi frontendom a backendom. Vždy keď databázová schéma je uložená alebo sa pripojí užívateľ, Laravel vyšle signál do fronty redis servera, ktorý potom odošle všetkým pripojeným klientom správu s dátami, ktoré sú nakonfigurované vo vysielaní v Laraveli. Jednotlivé cesty pre správu kolaborácie sú popísané nasledovne:

- `POST` /collaborations/{collaboration} - vytvorí novú kooperáciu s užívateľom
- `PUT` /collaborations/{collaboration} - upraví práva užívateľa v kooperácií
- `DELETE` /collaborations/{collaboration} - zruší kooperáciu s užívateľom

Kapitola 6

Testovanie

Pri testovaní som sa zameriaval hlavne na funkcionalitu a intuitívnosť užívateľského prostredia. Testovanie prebiehalo počas implementácie, kedy som skúšal jednotlivé vytvorené celky. Po kompletnej implementácií som zapojil do testovania aj dvoch kolegov z práce, ktorým bola vysvetlená funkcionalita a zadané úlohy popísané nižšie:

- Vytvoriť projekt typu kód a typu vstupných polí
- Vytvoriť databázovú štruktúru
- Exportovať do migračných súborov a modelov
- Otestovať vzájomnú kooperáciu

Jednotlivé ohlasy boli pozitívne, ale aj negatívne, ktoré apelovali na nedostatky. Poznámky na prehľadnosť celkového užívateľského prostredia boli veľmi pozitívne, nakoľko obaja vedeli bez akýchkoľvek doplňujúcich detailov pracovať s danou webovou aplikáciou. Ďalším pozitívnym ohlasom boli aj notifikácie, ktoré upozorňovali užívateľa na rôzne udalosti. Editor kódu bol veľmi pozitívne ohodnotený od oboch zúčastnených, keďže som k nemu pristupoval veľmi dôrazne.

Medzi nedostatky a postrehy v editore patrili chýbajúca možnosť premenovať projekt priamo v editore a hlavne prepisovanie už vytvorených databázových schém po uložení. Pri kooperácií taktiež bolo vytknuté, že užívateľ nedostane priamo notifikáciu alebo email o tom, že bol pozvaný do kooperácie. Ďalšou, ale menej dôležitou, poznámkou bolo zistenie pozvaného užívateľa, ktorí ostatní užívateľa sú ešte pripojení aktuálne v kooperácií, Nakoľko kooperácie môže spravovať iba hosťujúci užívateľ, táto možnosť je vypnutá pre ostatných. Posledným poznatkom bolo, nemožnosť voľne využívať editor diagramov a posúvať si vygenerované tabuľky, podľa vlastného uváženia.

Po testovaní som si zadal jasné ciele do budúcna, ktoré by som mal vylepšiť a prípadne pridať. Hlavným cieľom je vylepšiť spomínané prepisovanie databázových schém po uložení. v tomto prípade by bolo možné vytvoriť viacero verzií kódov, podobne ako pracuje git, alebo pridať možnosť povoliť zmeny, ktoré vykonal druhý užívateľ. Druhá možnosť by bola, ale z môjho pohľadu, viac nepraktická, keďže počet kolaborátorov, môže byť viac a mohlo by to byť veľmi nepraktické. Ďalším hlavným cieľom je spraviť editor diagramov, ktorý bude schopný presúvať a manažovať vytvorené diagramy, podľa vlastného uváženia.

V neposlednom rade by som chcel taktiež zamerať na celkový editor kódov, aby sa správal viac natívnejšie pre programátorov. Napríklad pridaním návrhov na doplnenie kódu alebo zvýraznenie riadku.

Kapitola 7

Záver

Cieľom práce bolo vytvoriť editor databázových schém s použitím pseudo-kódu alebo užívateľských vstupov. Hlavným zámerom tohto editora, bolo uľahčenie a urýchlenie procesu vytvárania databázových schém, ktoré sa pripravujú pre webové aplikácie s PHP aplikačným rámcom Laravel.

V práci bolo vysvetlené používanie aktuálnych webových technológií a použité technológie, ako napríklad frontendová knižnica React.js, práca s WebSocketmi pre spravovanie udalostí v reálnom čase a backendovým aplikačným rámcom Laravel. Celú prácu som začal analýzou všetkých potrebných technológií a porovnal som ich už s existujúcimi aplikáciami. Pri porovnávaní som si stanovil hlavné kritéria, ktoré by mali dané aplikácie spĺňať. Opísal som tri hlavné aplikácie, ktoré sa funkcionalitou približujú najviac k mojej práci. V konečnom dôsledku mala každá aplikácia svoje plusy a mínusy. Po analýze som pripravil návrh, ktorý obsahoval navrhnuté vylepšenia pre existujúce aplikácie. Začal som návrhom užívateľského prostredia, ktoré je veľmi dôležité v dnešnej dobe. Dobrý návrh užívateľského prostredia môže viesť k tomu, že o stránku bude mať viac ľudí záujem. Navrhoval som aj vlastný pseudo-kód, používaný pri písaní daných databázových schém. Tento pseudo-kód som sa snažil navrhnúť, čo najjednoduchšie, aby bol zrozumiteľný a zároveň dokázal vytvoriť databázové schémy podľa potreby. Po celkovom návrhu som začal s implementáciou webovej aplikácie. Implementáciu som opísal či už z pohľadu frontendu alebo backendu. Pri frontende som opisoval stránky, s ktorými užívateľ pracuje. Vysvetlil som taktiež proces vytvárania databázových schém pomocou obidvoch typov projektu. V backende bol opísaný celkový kompilátor pseudo-kódu, manažovanie projektu a generovanie súborov. Na neposlednom rade som spomenul, ako pracuje kolaborácia na backende a ako používam Redis spolu s Laravel Echo Server. Po implementácií som absolvoval testovanie s dvoma kolegami, ktorí mi pomohli pri získaní ďalších možných vylepšení a prípadných chýb. Po absolvovaní testovania som si ihneď rozšíril obzor, akým by sa táto práca mohla ďalej vylepšiť.

S výsledkom práce som veľmi spokojný, nakoľko mi rozšíril obzor v rôznych technológiách hlavne prácu s WebSocketmi, ktoré sú veľmi užitočné pri tvorení aplikácií v reálnom čase a myslím si, že to bol pre mňa štartom k využitiu tejto skvelej technológie aj pri iných projektoch. Taktiež som si rozšíril znalosti aj v oblasti frontendu s Reactom a aplikačným rámcom Laravel pri backende.

Literatúra

[1] Ahmed, R. *The Web Book Build Static & Dynamic Websites*. 1. vyd. Riaz Ahmed, 2013. ISBN 978-1483929279.

[2] Brennan, S. L. *A Simple Design Rule* [online]. 2017. 2022-10-27 [cit. 2023-04-23]. Dostupné z: https://www.saralynnbrennan.com/blog/the-60-30-10-design-rule.

[3] Budd, A. a Björklund, E. *CSS Mastery Advanced Web Standards Solutions*. 3. vyd. Apress, 2016. ISBN 978-1-4302-5863-6.

[4] Carlson, J. *Redis in Action*. 1. vyd. Manning Publications, 2013. ISBN 978-1617290855.

[5] Davidson, T. *Single Page Application (SPA) vs Multi Page Application (MPA): Which Is The Best?* [online]. 2023 [cit. 2023-05-01]. Dostupné z: https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/.

[6] Fette, I. a Melnikov, A. *The WebSocket Protocol* [online]. 2011 [cit. 2023-04-25]. DOI: 10.17487/RFC6455. ISSN 2070-1721. Dostupné z: https://www.rfc-editor.org/rfc/rfc6455.

[7] Gackenheimer, C. *Introduction to React*. 1. vyd. Apress, 2015. ISBN 978-1-4842-1245-5.

[8] IANA. *WebSocket Prtocol Registries* [online]. 2011. 2023 [cit. 2023-04-25]. Dostupné z: https://www.iana.org/assignments/websocket/websocket.xml#subprotocol-name.

[9] IANA. *HTTP Status Code Registry* [online]. 2022. 2022-06-08 [cit. 2023-04-18]. Dostupné z: https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml#http-status-codes-1.

[10] Lombardi, A. *WebSocket Lightweight Client-Server Communications*. 1. vyd. O'Reilly Media, Inc., 2015. ISBN 978-1-449-36927-9.

[11] Písek, S. *HTML - začínáme programovat*. 4. vyd. Grada Publishing, a.s., 2014. ISBN 978-80-247-5059-0.

[12] RedHat. *REST API* [online]. 2020 [cit. 2023-04-18]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[13] Robbins, J. N. *Learning Web Design*. 4. vyd. O'Reilly Media, Inc., 2012. ISBN 978-1-449-31927-4.

[14] Stauffer, M. *Laravel: Up & Running.* 1. vyd. O'Reilly Media, Inc., 2019. ISBN 978-1492041160.

[15] Welling, L. a Thomson, L. *PHP a MySQL: Kompletní průvodce vývojáře.* 1. vyd. Computer Press, 2017. ISBN 978-80-251-4892-1.

[16] Žára, O. *JavaScript Programátorské techniky a webové technologie.* 2. vyd. Computer Press, 2015. ISBN 978-80-251-4573-9.