Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Učebné prostredie pre online kolektívny vývoj agilných programov za pomoci zdieľateľného editoru

Bakalárska práca

2018
Emanuel Tesař

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Učebné prostredie pre online kolektívny vývoj agilných programov za pomoci zdieľateľného editoru

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. František Gyarfaš, CSc.

Bratislava, 2018
Emanuel Tesař

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

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

Názov: Učebné prostredie pre online kolektívny vývoj agilných programov za pomoci zdieľateľného editoru
Learning environment for online collaborative programming using sharable editor

Anotácia: Cieľom bakalárskej práce je vytvorenie web online editora pre kolektívne riešenie agilných úloh z programovania. Editor s kódom a testmi zdieľajú všetci účastníci skupiny, všetci môžu upravovať kód, pridávať testy, vyberať testy na zbiehanie a zbiehať ich vo virtuálnom prostredí na serveri. Aplikácia umožní pripojeným účastníkom zobrazenie kompilačných chýb, zobrazenie zbehnutých testov ako aj verzionovanie zdrojových súborov na serveri. Program umožní tvorbu skupín a ohodnocovanie úloh učiteľom. Súčasťou bude aj administratívne rozhranie pre zadávateľa úloh pre prípravu zadaní, viditeľné aj skryté testy, podľa ktorých sa automatizovane hodnotia výsledky. Technológie/nástroje: HTML 5, CSS, JavaScript (React), serverový framework (Express), Psql, virtuálny server.

Vedúci: Ing. František Gyarfaš, CSc.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 26.10.2018

Dátum schválenia: 06.11.2018 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Ďakujem môjmu školiteľovi Ing. Františkovi Gyarfašovi, CSc. za užitočné pripomienky, ochotu a usmernenie pri písaní diplomovej práce. Taktiež ďakujem svojej rodine a frejerke za ich pomoc a motiváciu.

Abstrakt

Cieľom práce je návrh a funkčná implementácia zdieľateľného editora podporujúca súbežné písanie kódu viacerých používateľov. Používatelia, okrem písania kódu, môžu pridávať testy, ktoré sa dajú spustiť vo virtuálnom prostredí na serveri. V prípade neskompilovateľného kódu, chyby počas behu programu, prípadne inej chyby, zobrazí chybovú hlášku. Okrem pohodlného prostredia pre používateľa obsahuje aj administrátorské rozhranie, v ktorom sa dajú pridávať neviditeľné testy pre používateľov. Toto rozhranie umožňuje profesorom vytvárať zadania pre študentov, ktorí následne môžu úlohu riešiť priamo vo webovom prehliadači.

Kľúčové slová: CRDT, kolaboratívne upravovanie, synchronizácia

Abstract

The aim of the work is to design and implement a shared editor supporting collaborative editing of multiple users. In addition to writing code, users can add tests that can run in a virtual server environment. In the case of uncompressed code, errors during program run, or other errors, will display an error message. In addition to a comfortable environment it also has an administrative interface for the user to add invisible tests for users. This interface allows professors to create assignments for students who can then solve the task directly in the web browser.

Keywords: CRDT, collaborative editing, synchronization

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

V súčasnosti je trendom vytvárať programy, ktoré sú zdieľateľné medzi používateľmi na viacerých počítačoch. Moderným prístupom je aj zdieľanie s možnosťou úprav, ktoré sa aplikujú v reálnom čase. Jedným z takýchto programov je aj editor kódu. Takéto programy už existujú, nie všetky však umožňujú daný kód zbiehať. Okrem iného náš program má mať možnosť vytvárať zadania, ktoré v nich používatelia môžu riešiť.

Takýto editor môže slúžiť ako učebná pomôcka pre učiteľov, ktorý vyučujú viacero kurzov. V našom programe dokážu vytvoriť zadanie pre svoje kurzy a do kurzov priradiť svojich študentov, ktorí môžu zadania riešiť. Takýto editor sa dá využiť na skupinových projektoch, prípadne predmetoch zaoberajúcich sa agilnými metodológiami ako párové programovanie.

Kapitola 1

Základné pojmy a definície

Latencia - reakčný čas označujúci dobu medzi akciou a následnou reakciou. Vo webových technológiách sa pod letenciou rozumie čas, medzi poslaním webovej požiadavky a následnej odpovede zo servera.

Dokument - zdroj, ktorý sa zdieľa medzi jednotlivými používateľmi

Replika - inštancia dokumentu, ktorá je zdieľaná medzi viacerými používateľmi. Keď nejaký používateľ zmení repliku, tak sa dané zmeny prejavia aj v ostatných replikách u iných používateľov.

Klient - klient predstavuje časť programu, ktorá je prístupná používateľovi. V tejto práci je klient jednostránková webová aplikácia.

Úplne usporiadaná množina - *taktiež známa pod názvom lineárne usporiadaná množina.* Úplne usporiadaná množina je usporiadaná množina, v ktorej sú každé dva rôzne prvky porovnateľné. Takouto reláciou je napríklad množina prirodzených čísel usporiadaná reláciou "menší ako".

Hustý priestor - vzťah na úplne usporiadaných množinách M, ktorý zabezpečuje, že pre ľubovolné $p, q \in M$ musí existovať nejaký prvok r, pričom platí $p \leq r$ a $r \leq q$.

Jednostránková webová aplikácia - typ webovej stránky, ktorý načíta zo servera jednu HTML stránku a pri interakcii so stránkou sa jej obsah mení dynamicky vo webovom prehliadači. Takéto stránky predstavujú moderný trend oproti klasickým statickým stránkam, ktoré pri interakciách žiadali server o nový obsah stránky, čo spôsobovalo dlhšie zobrazovanie.

Idempotencia - je vlastnosť algebraických operácií. Operácia je idempotentná, ak jej opakovaným použitím na nejaký vstup vznikne rovnaký výstup, ako vznikne jediným použitím danej operácie.

Optimistické zobrazovanie - využíva sa v prípade, že aplikácia zobrazuje nejaké entity a zároveň podporuje vytváranie nových. V optimistickom zobrazovaní používateľské rozhranie predpokladá, že odoslaná požiadavka na server prebehne úspešne. Dáta sú preto zobrazované hneď pri odoslaní požiadavky na server. To umožňuje vnímavejší zážitok používateľa, pretože nemusí čakať, kým sa zmeny prejavia aj na serveri.

NoSQL databáza - je databázový koncept, v ktorom dátové úložisko a spracovanie dát využíva iné prostriedky ako tabuľkové schémy tradičnej relačnej databázy. Databázy bez SQL sú často vysoko optimalizované pre typ kľúč-hodnota.

WebRTC - poskytuje okamžitú komunikáciu priamo medzi webovými prehliadačmi. Nie je nutný žiaden server, ktorý správy preposiela. Server je však nutný pri vytváraní komunikáčného kanálu medzi prehliadačmi.

Soket - je komunikačný kanál medzi klientom a serverom. Správy môžu byť cez soket posielané oboma smermi.

Kapitola 2

Kolaboratívny editor

Myšlienka kolaboratívneho editora bola prvýkrát zaznamenaná už v roku 1968 Douglasom Engelbartom. Do popularity sa však dostala až nedávno, približne 40 rokov od prvého záznamu. Kolaboratívny editor umožňuje viacerým používateľom naraz upravovať jeden dokument. Tieto editory sa rozdeľujú na dve kategórie podľa typu aplikovania úprav:

- v reálnom čase - zmeny dokumentu sa okamžite zobrazia všetkým používateľom
- s oneskorením - zmeny dokumentu sa nedejú okamžite (podobne ako pri verzionovacích systémoch ako Git, Mercurial)

V bakalárskej práci sa zaoberáme editormi so zmenami, ktoré aplikujú úpravy v reálnom čase, kde treba riešiť synchronizáciu editorových inštancií používateľov a riešenie možných konfliktov.

Problematika kolaboratívnych editorov v reálnom čase sa dá rozdeliť do samostatných zmysluplných podkapitol:

- technické výzvy
- algoritmy riešiace konkurentné modifikácie jedného dokumentu

2.1 Technické výzvy

Technické výzvy pramenia z asynchrónnej komunikácie po sieti. Teoreticky, keby táto komunikácia bola okamžitá, vytvorenie takéhoto editora by nebolo veľmi odlišné od editora pre jedného používateľa. Algoritmus, riešiaci takýto problém by mohol fungovať na základe *upravovacieho zámku*. Fungoval by celkom jednoducho:

1. Požiadanie servera o *upravovací zámok*
2. Počkanie na schválenie možnosti úpravy dokumentu zo servera

3. Úprava dokumentu

4. Vzdanie sa *upravovacieho zámku*

Keďže komunikácia so serverom je okamžitá, tak všetci používatelia by v čase modifikácie dokumentu mali rovnaký obsah dokumentu.

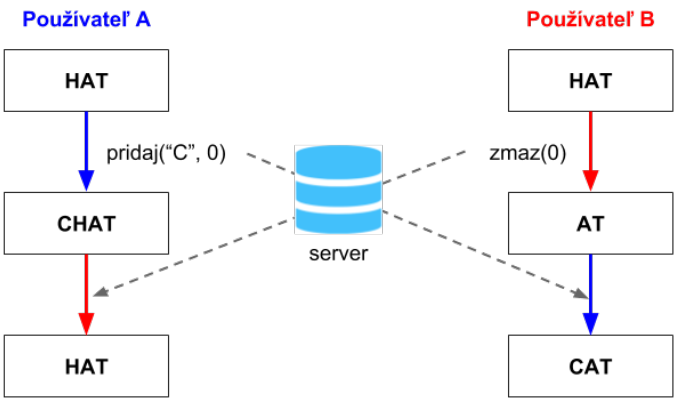
Avšak rýchlosť komunikácie je obmedzená latenciou siete. To vytvára základnú dilemu: používatelia potrebujú okamžite vidieť vlastné úpravy, ktoré sú do dokumentu zapracované, ale ak sú začlenené okamžite, tak pre latenciu komunikácie musia byť ich úpravy nevyhnutne vložené do rôznych verzií dokumentu.

Jednou s možností by mohlo byť zamedzenie konkurentných úprav dokumentu (algoritmus by bol podobný 2.1). To však vytvára zbytočné limitácie na architektúru riešenia, pretože treba vytvoriť mechanizmus, ktorý zabezpečuje, že dokument upravuje najviac jeden používateľ. Taktiež to zbytočne obmedzuje používateľov, pretože nemôžu naraz upravovať daný dokument.

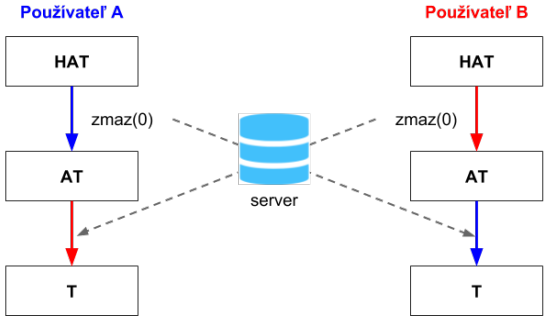
Ďalšou, pre používateľa príjemnejšou, voľbou je optimistická replikácia, kde sa zmeny v dokumente uplatňujú postupne a prípadné chyby a nekonzistentnosti sa riešia neskôr. Neskôr v texte ukážeme, že existuje spôsob, akým docieliť konkurentné úpravy dokumentu zaručene bez konfliktov. Takýto algoritmus je popísaný v kapitole 3.

Obmedzíme sa teraz iba na textový editor. Aj keď algoritmus, ktorý neskôr v texte ukážeme, je všeobecnejší a fungoval by aj pre iné typy editorov podporujúce zdieľanie iných entít, napríklad obrázkov, rôznych variantov textu a iných.

Problém súbežnej modifikácie jedného textového dokumentu je, že jednoduché textové operácie ako pridať alebo zmazať znak, nie sú komutatívne 2.1 a ani idempotentné 2.2. Keďže používatelia modifikujú dokument cez sieť, nie je zaručené, v akom poradí sa modifikácie uskutočnia. Ilustrujme tieto problémy na príklade:



Obr. 2.1: Nekomutativita textových operácii



Obr. 2.2: Neidempotentnosť textových operácii

Výzvou v spolupráci v reálnom čase je nájsť spôsob, akým možno aplikovať úpravy od vzdialených používateľov, ktoré boli pôvodne vytvorené vo verziách dokumentu, nikdy lokálne neexistovali a môžu byť v rozpore s vlastnými miestnymi úpravami používateľa.

Najsofistikovanejšie riešenia vyriešia tento problém spôsobom, ktorý nevyžaduje server, nepoužíva uzamknutie (všetci používatelia môžu voľne upravovať všetky časti dokumentu súčasne) a podporuje ľubovoľný počet používateľov (obmedzený iba zdrojmi počítačov). Tento prístup využíva napríklad *SubEthaEdit*, ktorý je však dostupný iba pre operačné systémy macOS a využíva technológie ako *Bonjour*, ktoré sú špecifické pre tento operačný systém. Viac sa o existujúcich programoch podporujúcich kolaboratívne úpravy dozvieme v kapitole 6.

Pri klient-server riešení je pri otvorení dokumentu priradená jednej z inštancií editora úloha servera. Tento server zaisťuje, že ostatné editory sú synchronizované. Pri každej zmene dokumentu inými používateľmi server obdrží upozornenia na vykonané zmeny, pričom zaistí, aby sa tieto zmeny aplikovali aj u ostatných používateľov. V niektorých modeloch sa zmeny na klientovi neodzrkadľujú dovtedy, kým sa zo servera nevráti odpoveď, že dané zmeny boli aplikované u všetkých používateľov. Príkladom takéhoto editora je editor *Gobby*. Tieto riešenia sú jednoduchšie na implementáciu, ale používateľov výrazne obmedzujú. Ak je latencia siete príliš vysoká, používateľ musí dlho čakať.

My sme sa v bakalárskej práci rozhodli použiť klient-server model, pričom za synchronizáciu klientov je zodpovedný výhradne server. Podobný prístup používa napríklad spoločnost Google vo viacerých svojich produktoch.

2.2 Algoritmy riešiace konkurentné modifikácie

Na riešenie synchronizácie klientov existujú dva dobre preskúmané typy algoritmov:

1. OT - Prevádzková transformácia *(angl. Operational transformation)*
2. CRDT - Bezkonfliktné replikovateľné dátové typy *(angl. Conflict-free replicated data type)*

V bakalárskej práci použijeme typ CRDT, pretože typ OT je teoretický model, ktorý predchádza typu CRDT a v praxi je jeho implementácia veľmi náročná a často nefunguje tak dobre, ako je spomínané v teórii. Taktiež použitie typu OT je vo väčších projektoch neškálovateľné [7].

Kapitola 3

Bezkonfliktné replikovateľné dátové typy

3.1 Úvod

Súbežné úpravy dokumentu bez koordinácie medzi jednotlivými používateľskými počítačmi môžu viesť k nekonzistentnostiam, ktoré vo všeobecnosti nie je možné vyriešiť. Na zabezpečenie konzistentnosti sa v takýchto prípadoch vyžaduje vrátenie *(angl. roll-back)* niektorých operácií.

V distribuovanom výpočte je bezkonfliktný replikovaný dátový typ (CRDT) dátová štruktúra, ktorá môže byť replikovaná vo viacerých počítačoch v sieti. Jednotlivé inštancie je možné aktualizovať nezávisle a súbežne bez koordinácie medzi ostatnými inštanciami, pričom je vždy matematicky možné vyriešiť nezrovnalosti, ktoré by mohli nastať. Z toho vyplýva, že použitím typu CRDT nemôže nastať situácia, kedy by sme museli nejakú operáciu vrátiť.

Koncept CRDT formálne definovali v roku 2011 osoby Marc Shapiro, Nuno Preguiça, Carlos Baquero a Marek Zawirski [10]. Ide teda o celkom nový koncept, ktorý rýchlo našiel v informatike podporu.

3.2 Typy CRDT

V súčasnosti sú preskúmané 2 typy CRDT:

1. CRDT založené na operáciách
2. CRDT založené na stavoch

3.2.1 CRDT založené na operáciách

CRDT založené na operáciách sú označované ako komutatívne replikované dátové typy alebo *CmRDT*. CmRDT menia dokument vysielaním iba operácií aplikovaných lokálne u všetkých používateľov, ktorí dokument zdieľajú. Operácie musia byť komutatívne, avšak nemusia byť idempotentné. Je preto nutné zaručiť idempotenciu pomocou vhodnej implementácie komunikácie (správa musí byť vyslaná práve raz). CmRDT je vhodné použiť napríklad na aktualizovanie počtu zdieľaní príspevkov na sociálnych sieťach, kde server zaručuje idempotentnosť.

3.2.2 CRDT založené na stavoch

CRDT založené na stavoch sa nazývajú konvergentné replikované dátové typy alebo *CvRDT*. Na rozdiel od CmRDT posielajú celý svoj stav, kde stavy sa potom zlučujú funkciou, ktorá musí byť komutatívna, asociatívna a idempotentná.

Posielanie celého stavu nie je vždy optimálne. Častokrát stačí posielať zmeny, ktoré sa na dokumente uskutočňujú. Takéto CRDT označujeme ako *Delta CRDTs*

3.3 Princíp fungovania CRDT

CRDT funguje tak, že každý znak v dokumente sa prerobí do objektu so špecifickými vlastnosťami:

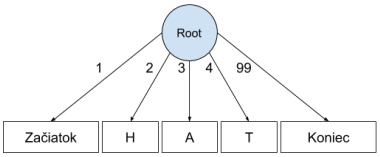
1. znak, ktorý objekt predstavuje
2. relatívna pozícia tohto znaku
3. množina pozícií musí tvoriť úplné usporiadanie
4. priestor tvorený množinou pozícií musí byť hustý

Vzhľadom na to, že každý z týchto objektov je jedinečný a môže byť identifikovaný týmito vlastnosťami, môžeme zabrániť vloženiu alebo vymazaniu znakov viac ako raz. To umožňuje komutativitu a idempotenciu. Nevýhodou tohto prístupu je veľké množstvo metaúdajov. Tým sa zvyšuje spotreba pamäte aplikácie.

Zaujímavým aspektom CRDT, ktorý ho odlišuje od OT, sú relatívne pozície znakov. Tieto pozície majú nasledovné vlastnosti:

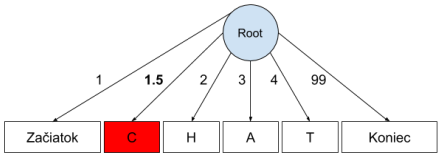
1. žiadne dva CRDT objekty nemajú rovnakú pozíciu
2. pozícia nejakého objektu sa nikdy nezmení
3. ak dva CRDT objekty A a B operujú na tej istej pozícii v dokumente a objekt A nastane skôr než B, tak relatívna pozícia A musí byť menšia ako B.

Vytvorenie takýchto pozícií je celkom jednoduché. Znaky si môžeme predstaviť ako vrcholy na strome, kde každý znak má väčsie číslo ako znak pred ním, no menšie ako znak po ňom. Pozície môžu vyzerať zhruba nasledovne:



Obr. 3.1: Relatívne pozície znakov

Pridanie znaku potom funguje veľmi jednoducho. Nájdeme v strome vrcholy, medzi ktoré chceme pridať ďalší znak a ako pozíciu mu dáme priemer relatívnych pozícií daných vrcholov. Napríklad:



Obr. 3.2: Pridanie relatívnej pozície znakov

Komutativita a idempotentnosť CRDT

Ak predpokladáme, že pozície spĺňajú podmienky z 3.3, tak CRDT objekty sú komutatívne a idempotentné.

Idempotencia je daná tým, že ak chceme do dokumentu pridať znak s nejakou relatívnou pozíciou, a tá sa tam už nachádza, tak opätovné pridanie už nič neurobí. Ak sa naopak snažíme vymazať niečo na pozícii, ktorá sa v dokumente už nenachádza, tak vieme, že uz vymazaná bola. Obe operácie teda zachovávajú idempotentnosť. [9]

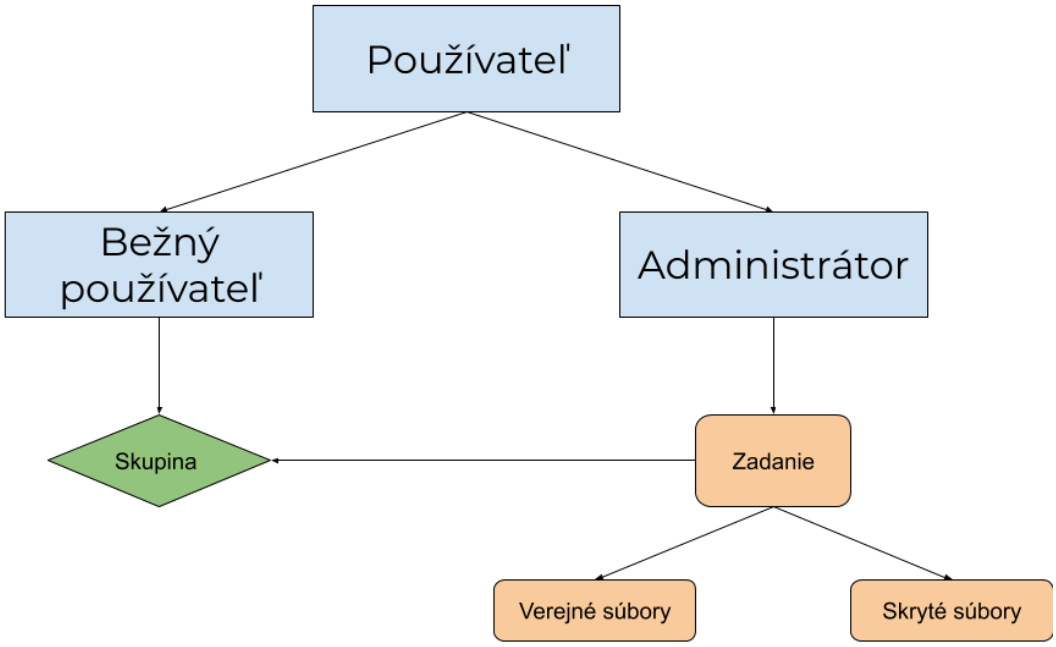
3.4 Použitie v praxi

Aplikácie CRDT sa dajú nájsť nielen v oblasti kolaboratívnych editorov, ale na mnohých ďalších miestach ako napríklad databáza *Redis*, platforma na zdieľanie hudby *SoundClound* alebo NoSQL dátové úložisko *Riak*, ktoré sa používa na vnútorný četovací systém v hre *League of Legends*.

Kapitola 4

Návrh programu

V návrhu bakalárskej práce vystupujú rôzne entity, medzi ktorými sú rôzne vzťahy.



Obr. 4.1: Entity v návrhu

Používateľ - Entita pomenúvajúca používateľa aplikácie. Používateľov rozdeľujeme na bežných používateľov a administrátorov.

Bežný používateľ - Bežní používatelia predstavujú zovšeobecnený pojem pre študentov. Každý bežný používateľ patrí do niekoľkých skupín, v ktorých má prístup iba k úlohám prístupným pre danú skupinu.

Administrátor - Administrátor má plný prístup ku všetkým informáciám na serveri. Okrem toho pridáva bežných používateľov do skupín, pridáva zadania, vidí odovzdané zadania bežných používateľov...

Skupina - Skupina je entita, ktorá sa skladá z bežných používateľov. Každé zadanie musí patriť do práve jednej skupiny. Toto zadanie je prístupné iba pre používateľov, ktorí patria do danej skupiny.

Zadanie - Administrátor vie vytvoriť zadanie pre nejakú skupinu, ktoré môžu bežní používatelia riešiť. Zadanie obsahuje verejné a skryté súbory.

Verejné súbory - Verejné súbory sú súbory zadania, ktoré sú prístupné pre používateľa.

Skryté súbory - Skryté súbory by nikdy nemali byť posielané na klienta, a slúžia na vytvorenie skrytých vstupov a správnych riešení, prípadne iných súborov, ku ktorým by klient nemal mať prístup.

Program sa skladá z viacerých častí, ktorých návrhy popíšeme v tejto kapitole. Aplikácia je založená na klient-server architektúre. Program sa preto dá rozdeliť na podkapitoly *klient* a *server*.

4.1 Klient

Klient je v bakalárskej práci webová aplikácia, cez ktorú používateľ komunikuje so serverom. V návrhu klienta sa snažíme hlavne vytvoriť jednoduché používateľské prostredie, ktoré je intuitívne na použitie. Kedže aplikácia má byť používaná ako editor, je nutné, aby sa jednotlivé časti pohodlne čítali a písmo bolo dostatočne veľké. Naopak, nepredpokladáme použitie editora na mobilných zariadeniach. Klientská aplikácia obsahuje:

- rozhranie pre prihlasovanie a registráciu
- administrátorské rozhranie
- rozhranie pre bežného používateľa

4.1.1 Prihlasovanie a registrácia

Prihlasovanie slúži na jednoduché kategorizovanie používateľov a na prístup k editoru. Každý používateľ má svoje prihlasovacie meno a heslo, pod ktorým sa zaregistroval. Prihlasovanie a registrácia sú pre oba typy používateľov rovnaké.

Registrácia umožňuje vytvoriť nového používateľa. Zaujímavá otázka je, či umožniť vytváranie administrátorov priamo pri registrácii, alebo nejakým iným spôsobom. Rozhodli sme sa, že je lepšie, ak používateľ nemusí pri registrácii riešiť koncept administrátorov, a teda umožniť iba vytváranie bežných používateľov.

4.1.2 Administrátorské rozhranie

Po prihlásení sa administrátorovi automaticky zobrazí administrátorské rozhranie, ktoré je rozdielne od rozhrania pre bežného používateľa. Adminstrátorské rozhranie umožňuje:

- zobraziť, upraviť a odstrániť používateľa
- zobraziť všetky uložené zadania
- zobraziť všetky odovzdané zadania
- vytvoriť, upraviť a odstrániť zadanie
- pridať, upraviť a odstrániť skupiny

4.1.3 Rozhranie pre bežného používateľa

Rozhranie pre bežného používateľa slúži iba nato, aby si používateľ mohol vybrať skupinu a zadanie, na ktorom chce pracovať. Po zvolení zadania sa zobrazí editor, v ktorom môže používateľ zadanie riešiť. Editor sa automaticky synchronizuje na vzdialených počítačoch, prípadne nových kartách vo webovom prehliadači. Synchronizácia je granulovaná pre používateľa. To znamená, že ak sa dva počítače navzájom synchronizujú, sú práve prihlásené pod tým istým používateľom.

Panel so súbormi

Úlohy sa môžu skladať z viacerých súborov. Napríklad môže existovať súbor, ktorý obsahuje zadanie a súbor (prípadne viac súborov) na zdrojový kód. Panel so súbormi dovoľuje používateľovi pohodlne preklikávať a zvoliť aktívny súbor, ktorý sa zobrazuje v editore.

Zdieľateľný editor

Najzaujímavejšou častou práce je kolaboratívny editor umožnujúci bežným používateľom naraz upravovať kód. V editore musí byť jasné, ktorý súbor sa práve upravuje. Editor môže tiež ponúkať nejaké informácie o vzdialených používateľoch ako prihlasovacie meno, pozícia kurzora a ďalších.

Ovládací panel
Samotná úprava kódu je dôležitá, ale okrem toho je potrebné verzionovanie súborov, teda nejaké ukladanie a načítavanie. Okrem toho by používateľ mal mať možnost práve upravovaný kód spúštať na vlastnom vstupe alebo na skrytých vstupoch danej úlohy.

4.2 Server

Server slúži na komunikáciu s klientom a obsahuje všetky potrebné informácie o bežných používateľoch, administrátoroch, skupinách a vytvorených zadaniach.

Server má viacero funkcií:

- synchronizácia replík klientov
- uchovávať informácie o používateľoch a zadaniach
- poskytovať API pre klienta
- zbiehanie kódu, poskytovanie odpovede klientom

4.2.1 Synchronizácia replík klientov

Najdôležitejšou funckiou servera je synchronizácia replík klientov, ak nastane nejaká zmena dokumentu. Server na toto nepotrebuje veľkú logiku, ale potrebuje korektne preposielať informácie medzi klientami a reagovať na prípadné komunikačné chyby (odoslať správu znovu).

4.2.2 Uchovávať informácie o používateľoch a zadaniach

Po registrácii sa musia na serveri uložiť informácie o používateľovi, aby sa následne mohol opätovne prihlasovať. Okrem toho klient môže ukladať a načítavať zadania, ktoré tiež musia byť nejakým spôsobom na serveri uložené.

4.2.3 Poskytovať API pre klienta

Hlavnou funckiou servera je poskytovať API rozhranie, ktoré môže klient zavolať a vykonať nejakú akciu na serveri. Časť rozhrania sluzi na získanie informácií zo servera a časť vytvára na serveri nové položky (používatelia, zadania).

4.2.4 Zbiehanie kódu, poskytovanie odpovede klientom

Kód, ktorý sa napíše na klientoch, sa má dať spustiť v izolovanom prostredí servera. Aplikácia by mala poskytnúť dve možnosti ako testovať klientský kód:

- na vlastnom vstupe
- na skrytých vstupoch

Pri testovaní na vlastnom vstupe by používateľ na klientovi mal mať možnosť zadať vstup a potom odoslať požiadavku na testovanie serveru. Po otestovaní by sa mal klientovi zobraziť výsledok testovania. Testovanie na vlastných vstupoch nemusí byť ukladané na serveri, stačí ak server na požiadavku odpovie. Ak by výsledok testovania skončil chybou pri kompilácii, prípadne chybou počas behu programu, daná chybová hláška sa má zobraziť ako výstup programu.

Okrem testovania na vlastných vstupoch si vie používateľ otestovať kód aj na skrytých vstupoch a výstupoch zadania. Takéto skryté vstupy vie pridať iba administrátor a nikdy by nemali byť zobrazené bežnému používateľovi. Každé testovanie na skrytých vstupoch musí byť uložené na serveri, aby si ho administrátor vedel pozrieť, keď bude ohodnocovať dané zadanie bežného používateľa. Po odoslaní požiadavky serveru na otestovanie na skrytých vstupoch by sa mali zobraziť výsledky testovania. Skrytých vstupov môže byť viac, ale testuje sa na vstupoch zaradom a keď sa nájde vstup, na ktorom program dáva nesprávny výstup, tak sa v testovaní ďalej nepokračuje. Výsledok testovania jedného vstupu môže byť:

- OK - Kompilácia a aj beh programu prebehli v poriadku a výsledok programu na danom vstupe je totožný so správnym výstupom
- WA - Kompilácia prebehla v poriadku a program úspešne skončil, ale výstup sa nezhoduje so správnym výstupom
- TLE - Kompilácia prebehla v poriadku, ale program prekročil časový limit, ktorý je dostupný pre dané zadanie.
- RTE - Kompilácia prebehla v poriadku, ale program počas behu programu spadol.
- CE - Kompilácia programu sa nepodarila.

(V prípade, že jazyk, v ktorom je kód napísaný, je interpretovaný, sa kompilácia preskakuje. Stále však môžu nastať všetky výsledky testovania s výnimkou CE)

Kapitola 5

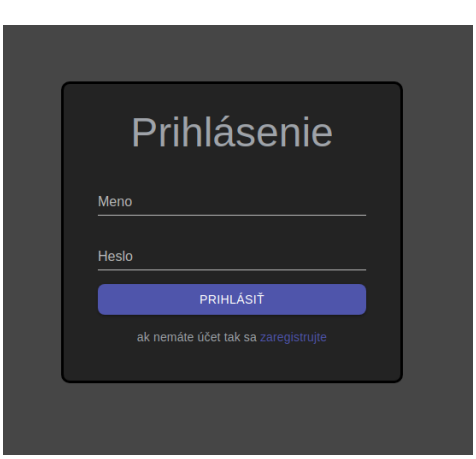
Implementácia

V predošlej kapitole sme popísali návrh aplikácie, ale nevenovali sme sa implementačným detailom. Implementácia klienta a servera je odlišná. Na klientovi riešime implementačné detaily vzhľadu stránky, zatiaľ čo na serveri štruktúru entít a ich ukladanie v databáze.

5.0.1 Klient

Klient je implementovaný ako jednostránková webová aplikácia. Celý klient je napísaný v knižnici *React* a jazyku *Typescript*. Typescript je nadmnožinou jazyka JavaScript, ktorá do tohto dynamického a netypovaného jazyka pridáva statické typy. Vďaka tomu je kód menej náchylný na chyby a výhodou je tiež automatické navrhovanie kódu, ktoré editor poskytuje, pretože vie, čo je akého typu. Komponenty zobrazované na klientovi využívajú knižnicu *Material-UI*, ktorá obsahuje jednoduché komponenty, ktoré sú naštýlované poďla dizajnov, ktoré používa spoločnosť Google.

Pri načítaní stránky sa používateľovi zobrazí prihlasovacie okno, cez ktoré sa má do aplikácie prihlásiť. Ak používateľ ešte účet nemá, vie sa prekliknúť na registráciu.

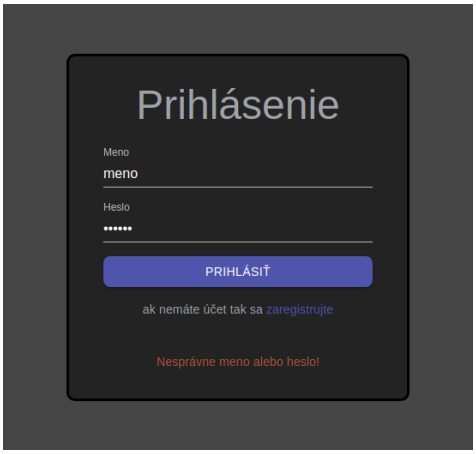


(a) Prihlásenie

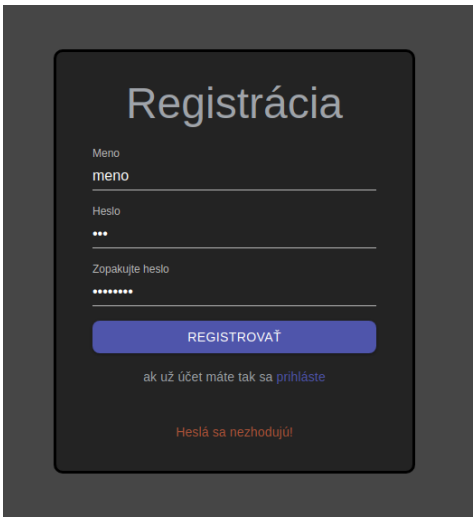
(b) Registrácia

Obr. 5.1: Prihlásenie a registrácia

Prihlasovací formulár validuje, či je používateľ zaregistrovaný, a či zadal správne heslo. Podobne registrovací formulár kontroluje, či je používateľské meno voľné, a či sa heslá zhodujú. Prípadné chyby zobrazuje pod formulárom.



(a) Validácia prihlásenia



(b) Validácia registrácie

Obr. 5.2: Validácia prihlásenia a registrácie

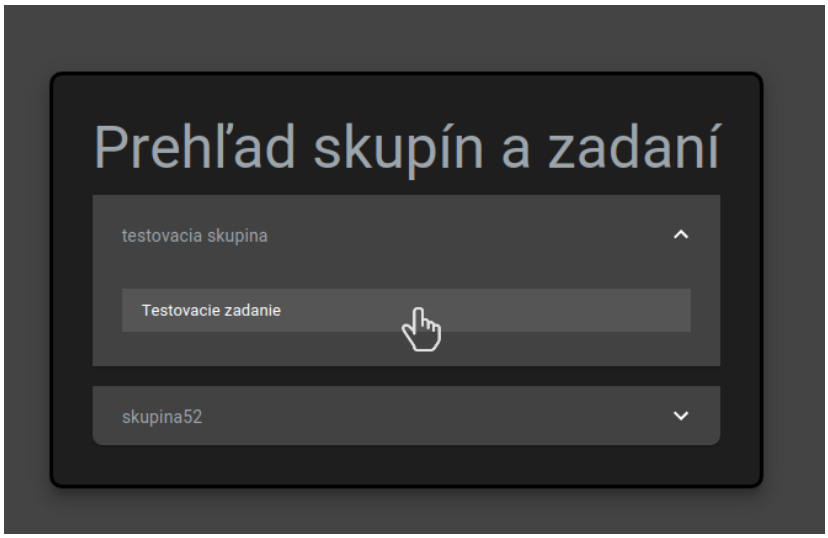
Po prihlásení používateľa sa zobrazí hlavná časť programu. Odlišuje sa však výrazne podľa toho, či je prihlásený administrátor alebo bežný používateľ.

5.0.2 Rozhranie pre bežného používateľa

Pri implementácii rozhrania pre bežného používateľa sme použili kombináciu odtieňov sivej na pozadia a farby editoru a bielu farbu na písmo. Na výraznenie akcií, ktoré

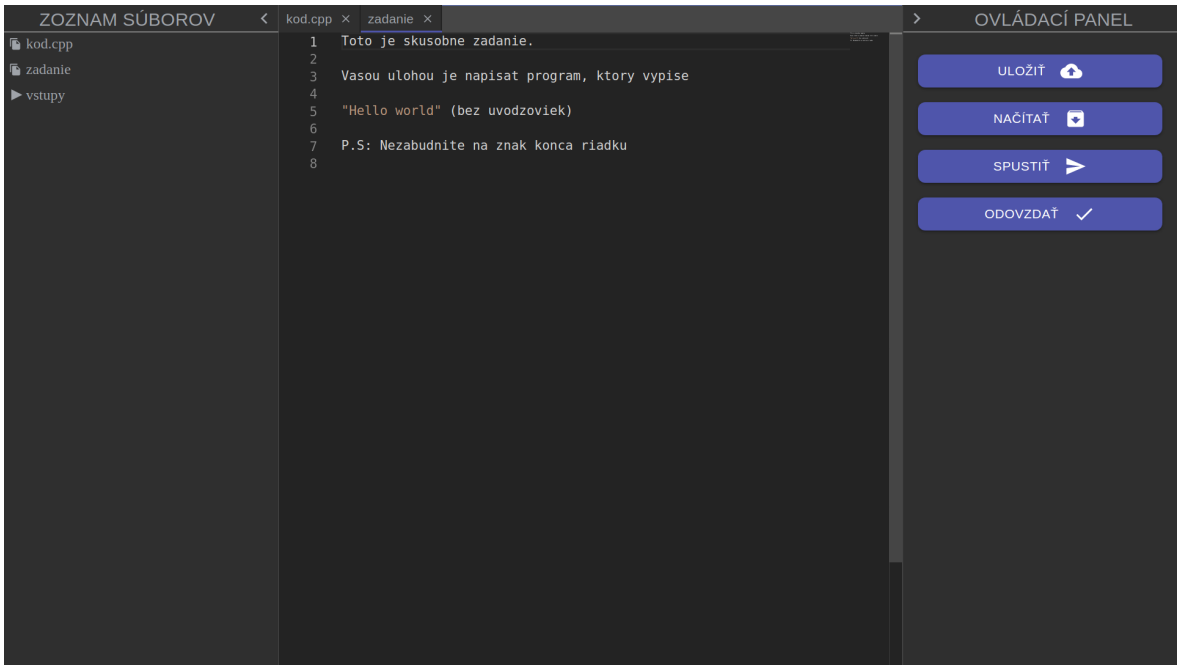
vie používateľ na stránke robiť, používame špecifický odtieň modrej farby. Poslednou používanou farbou je červená, ktorou signalizujeme chyby.

Po prihlásení sa zobrazí bežnému používateľovi rozhranie, v ktorom vidí všetky vytvorené skupiny. Nevidí však ani ich obsah, ani prihlásených používateľov patriacich do daných skupín. Do skupín ho vie pridať iba administrátor. Ďalej sú v rozhraní zobrazené skupiny, do ktorých používateľ patrí a v nich vidí aj ostatných používateľov spolu so zadaniami pre danú skupinu.



Obr. 5.3: Výber zadania

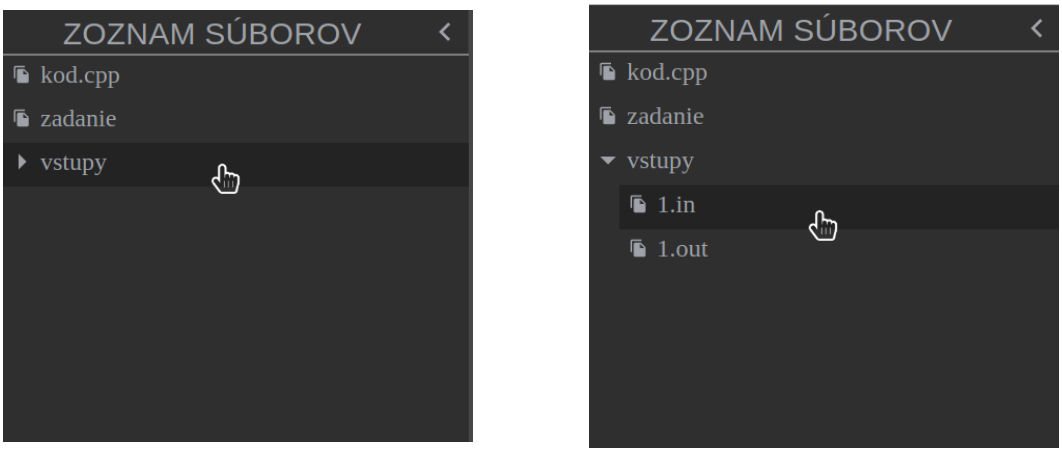
Ak si používateľ vybral zadanie, otvorí sa mu editor, v ktorom vie upravovať všetky súbory dostupné pre dané zadanie. Súčasťou editora sú viaceré komponenty, ktoré používateľovi uľahčujú a sprehľadňujú prácu s editorom.



Obr. 5.4: Editor zadania

Panel so zoznamom súborov

Komponent, ktorý obsahuje hierarchickú schému priečinkov a súborov dostupných v zadaní. Priečinky sa dajú otvárať podobne ako pri otváraní súboru na počítači. Súbor alebo priečinok, na ktorý ukazuje kurzor sa zvýrazní a kurzor sa zmení na ukazateľ, aby používateľ vedel, že môže na daný súbor alebo priečinok kliknúť. Ikonky jasne odlišujú súbor od priečinku. Priečinok môže byť v schovanom alebo zobrazenom stave. Tieto stavy sú rozlíšené typom ikonky.



(a) Schovaný obsah priečinka

(b) Zobrazný obsah priečinka

Obr. 5.5: Priečinok v zozname súborov

Ovládací panel

Ovládací panel umožňuje používateľovi vykonávať akcie so svojím zadaním. Používateľ

si môže svoj kód uložiť, načítať, spustiť na vlastnom vstupe alebo otestovať na skrytých vstupoch tak, ako to bolo napísané v návhu. Pri kliknutí na ľubovolnú akciu sa zobrazí dialóg, v ktorom používateľ vyplní údaje a potvrdí akciu, ktorá sa následne odošle na server.

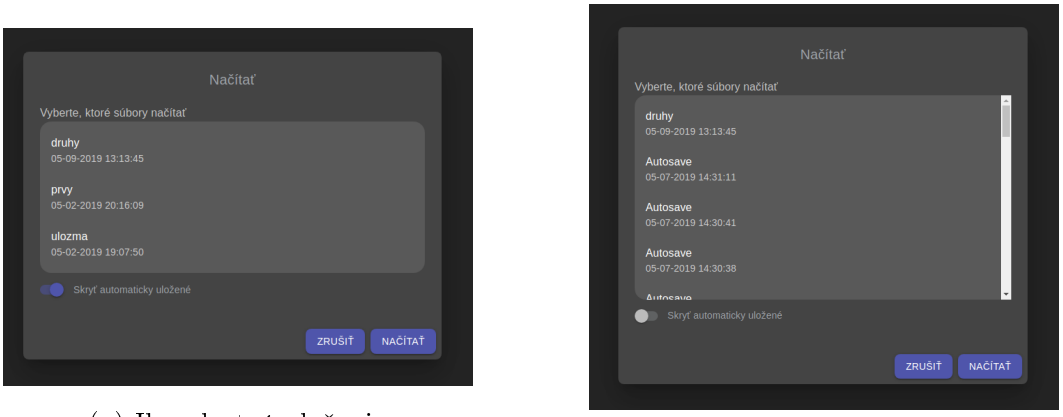
Pri ukladaní zadania je nutné zvoliť identifikátor, pod ktorým sa má zadanie uložiť. Identifikátor môže obsahovať ľubovolné znaky, no nesmie byť prázdny. Ak používateľ zadá neplatný identifikátor, klient mu súbor uložiť nedovolí.

(a) Neplatný názov identifikátora

(b) Platný názov identifikátora

Obr. 5.6: Uloženie zadania

Pri načítavaní uloženého zadania sa používateľovi zobrazia všetky jemu dostupné uložené zadania v zozname. Používateľ si vie vybrať, ktorý záznam chce načítať poďla identifikátora, ktorý zadával pri ukladaní a časovej pečiatky, kedy záznam uložil. Pri spúštaní a odovzdávaní zadania aplikácia sama od seba uloží obsah zadania. Používateľ si vie zvoliť, či chce takéto záznamy vidieť v možnostiach.



(a) Iba vlastné uloženia

(b) Zobrazené aj automaticky uložené

Obr. 5.7: Načítanie zadania

Po spustení sa zobrazí dialóg, v ktorom vie používateľ zadať vlastný vstup a následne spustiť svoj program na danom vstupe. Používateľ potom v dialógu potvrdí spustenie a program sa spolu so vstupom, ktorý zadal, odošle na server, kde sa momentálny stav zadania automaticky uloží, skompiluje a následne spustí. Počas behu programu sa v dialógu zmení tlačítko, ktoré upozorňuje, že program beží. Po skončení behu programu sa zobrazí výsledný výstup v dialógu.

(a) Pred spustením (b) Počas behu (c) Po dokončení

Obr. 5.8: Spúšťanie na vlastnom vstupe

Posledná akcia, ktorú vie používateľ spraviť, je otestovanie svojho riešenia na skrytých vstupoch uložených na serveri. Po kliknutí na tlačítko sa odošle na server aktuálny stav zadania, ktorý sa na serveri uloží a automaticky začne testovať.

(a) Dialóg počas testovania

(b) Dialóg po skončení testovania

Obr. 5.9: Testovanie na skrytých vstupoch

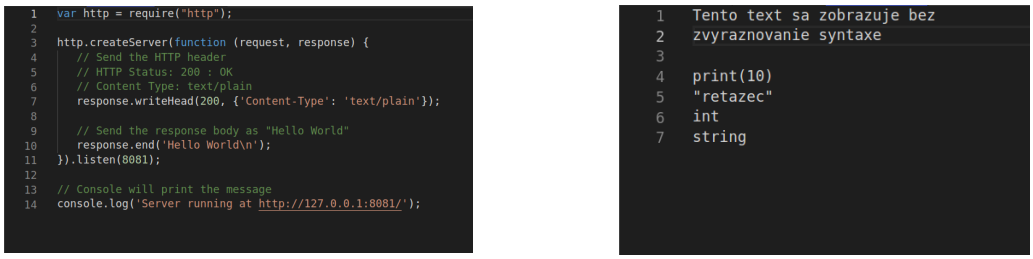
Panel s otvorenými súbormi

Dôležité je, aby používateľ vedel efektívne preklikávať medzi otvorenými súbormi a vedel, ktorý súbor práve upravuje. Nato slúži horný panel editora, v ktorom sú zobrazené karty s práve otvorenými súbormi. Aktívny súbor je podčiarknutý modrou farbou, aby bolo zvýraznené, ktorý súbor sa práve upravuje. Súbory sa v hornom paneli dajú zavrieť v pravej časti karty klikom na krížik. Pri posune kurzora nad oblasť krížika sa krížik zväčší, čím upozorní používateľa, že kliknutím zavrie danú kartu.

Editor kódu

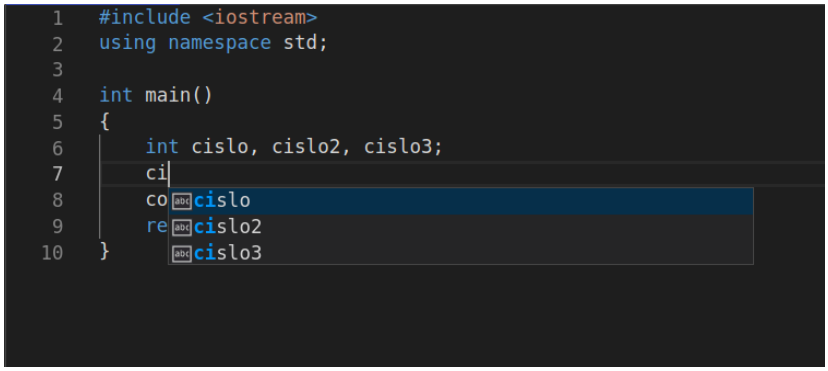
Najdôležitejším komponentom je samotný editor, v ktorom sa dá písať zdrojový kód. Editor použitý v práci sa nazýva *Monaco*, ktorý je použitý aj v populárnom editore *VS Code*.

Monaco automaticky zvýrazňuje syntax zdrojového kódu v práve otvorenom súbore. To, v ktorom jazyku je zdrojový kód napísaný, editor rozpozná podľa koncovky súboru. Monaco vie zobraziť syntax pre všetky bežné programovacie jazyky, pričom sa dajú pridať ďalšie. V prípade, že sa editoru nepodarí rozpoznať jazyk, v ktorom je obsah súboru napísaný, bude obsah zobrazený bez špeciálneho zvýraznenia syntaxe.



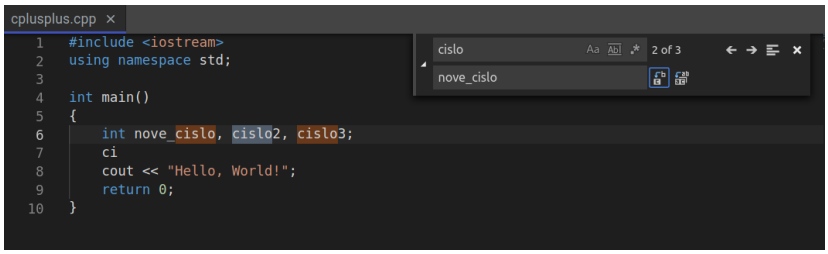
Obr. 5.10: Zvýrazňovanie syntaxe jazyka

Okrem zvýrazňovania syntaxe editor poskytuje základné operácie ako krok späť a krok dopredu. Pre jednoduchú navigáciu v texte je v pravej časti editoru "minimapka". Avšak najužitočnejšou a najviac používanou funkcionalitou editora je nepochybne automatický návrh. Ak používateľ napíše pár znakov, editor mu ponúkne na výber z možností, ktoré by mohol chcieť napísať. Používateľ môže zobraziť návrh aj stlačením klávesovej skratky *Ctrl + medzera*. Do návrhu sa postupne pridávajú slová, ktoré používateľ do editoru napíše. V zozname navrhovaných možností sa používateľ pohybuje šípkami a možnosť zvolí stlačením klávesy *Enter*.



Obr. 5.11: Automatický návrh

Pri programovaní je dôležité vedieť v kóde nájsť nejaký reťazec, prípadne zmeniť nájdené výsledky na nejaký iný reťazec. Monaco túto funkcionalitu podporuje po stlačení klávesovej skratky *Ctrl + F* pre hľadanie a *Ctrl + H* pre hľadanie spolu s nahradzovaním. Obe možnosti podporujú rôzne typy vyhľadávania, ako napríklad hľadanie pomocou regulárnych výrazov. Na postupné označovanie nejakého reťazca sa dá použiť aj multikurzor. Stačí, ak používateľ označí text a stlačí *Ctrl + D*. Tento príkaz nájde a označí ďalší výskyt s rovnakým textom, aký používateľ označil.

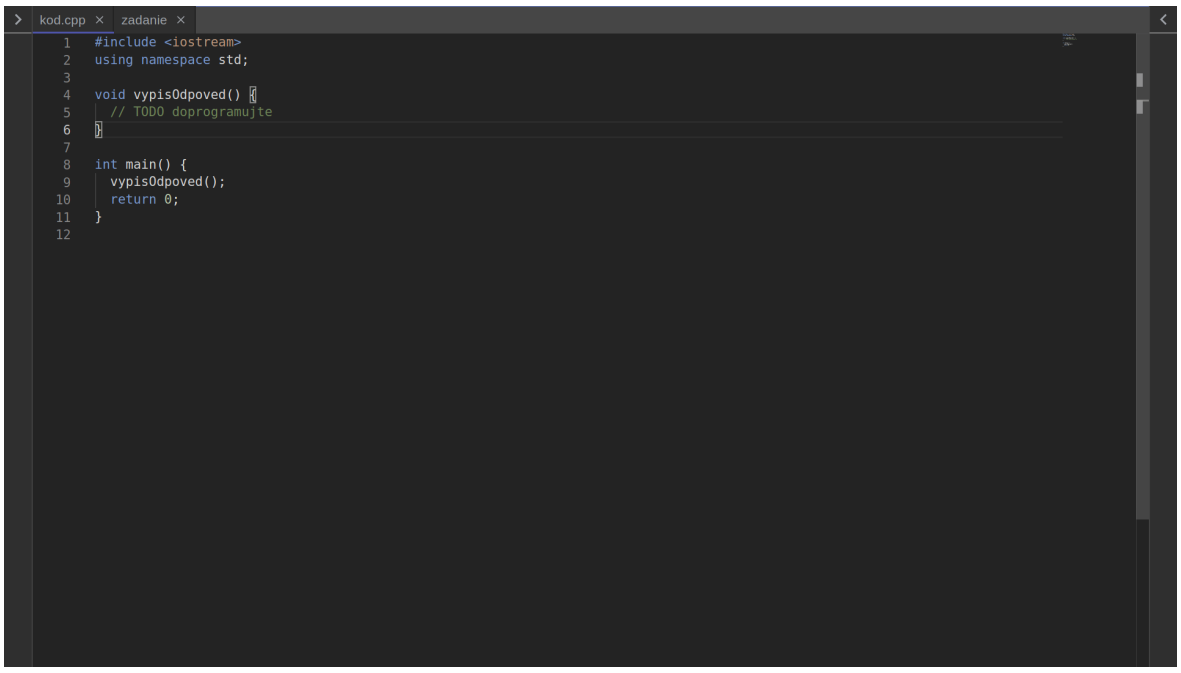


Obr. 5.12: Hľadanie a nahradzovanie

Monaco obsahuje aj mnoho ďalších funkcií. Kompletný zoznam sa dá zobraziť stlačením klávesy *F1*.

Schované panely

Oba panely zaberajú značný horizontálny priestor a nie vždy sú pre používateľa nutné. Používateľ môže panely v editore schovať, čím rozšíri priestor určený pre editor. Používateľ môže kedykoľvek panel zobraziť naspäť kliknutím na šípku, ktorá panel rozvinie a editor sa znovu zmenší.



Obr. 5.13: Editor zadania so schovanými panelmi

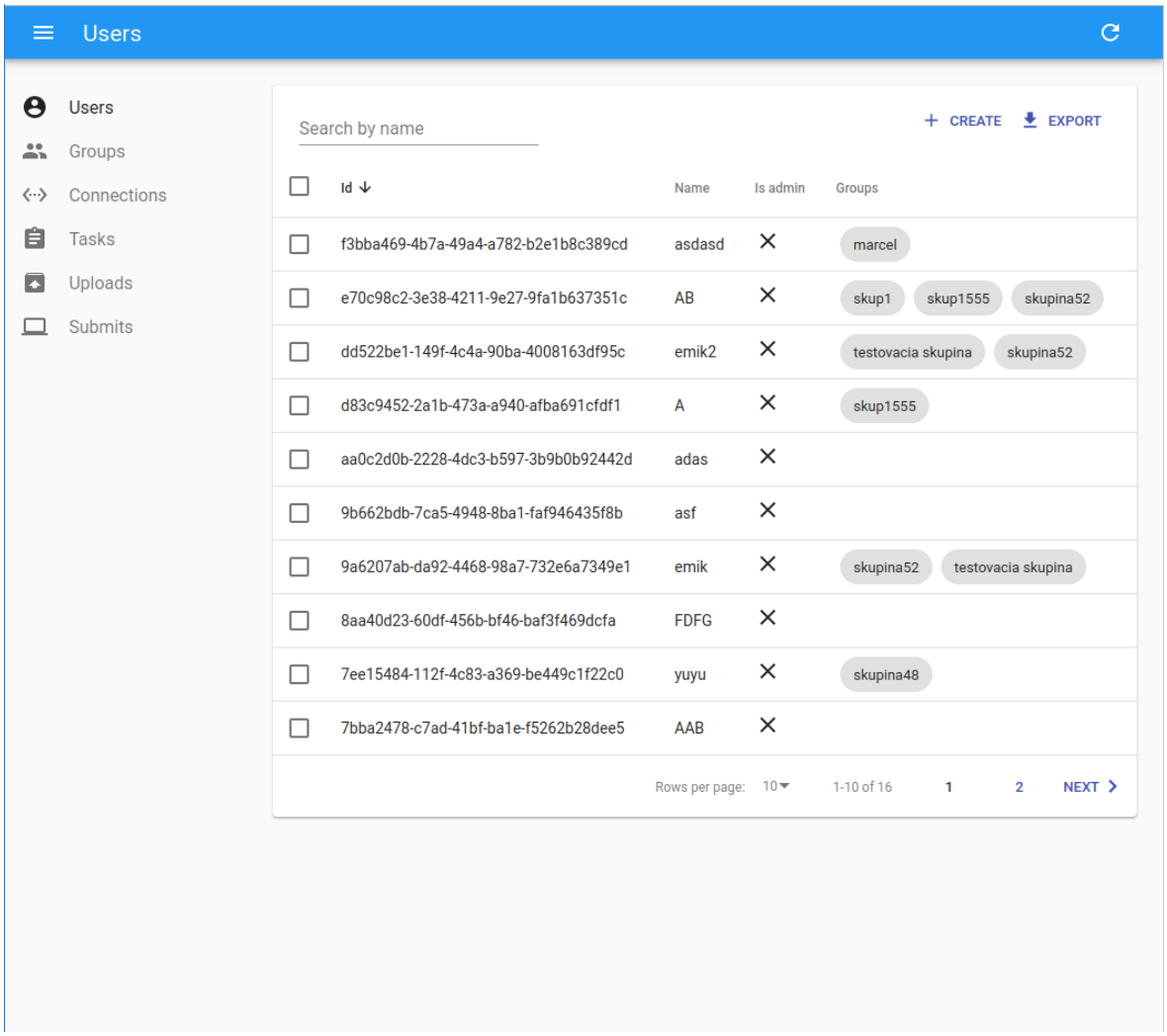
5.0.3 Rozhranie pre administrátora

V návrhu aplikácie sme povolili registráciu iba pre bežného používateľa. Zároveň sme v aplikácii neumožnili žiaden spôsob, ako spraviť z bežného používateľa administrátora. Preto vytvorenie administrátora v klientskej aplikácii nie je možné. Administrátor sa dá vytvoriť iba pridaním administrátorského účtu priamo do databázy na serveri. Keď

sa administrátor prihlási na klienta, zobrazí sa mu automaticky administrátorské rozhranie.

Farby použité na rozhranie pre adminsitrátora sú úplne iné oproti farbám zvoleným na rozhranie pre bežného používateľa. Administrátorské rozhranie používa na pozadie bielu farbu a text používa farby odtieňov sivej. Tieto farby sú predvolenými farbami knižnice *React admin*, ktorá je použitá na vytvorenie rozhrania. Jazyk použitý v rozhraní je anglický, pretože knižnica používa anglický jazyk a preklad do slovenčiny by bol náročný a zdĺhavý.

React admin dovoľuje deklaratívne popísať administrátorské rozhranie a jediná požiadavka knižnice je webová cesta, na ktorej je dostupný server. Aplikácia potom volá rozhranie servera a výsledné dáta zobrazuje. Knižnica je veľmi užitočná, lebo sa dá použíť s ľubovolným typom servera, tiež používa *Material-UI* a využíva optimistické zobrazovanie. Veľkým bonusom je aj detailná dokumentácia a aktívna komunita podporujúca túto knižnicu.



Obr. 5.14: Administrátorské rozhranie

Administrátorské rozhranie obsahuje navigačný panel, ktorý rozhoduje o tom, ktoré entity sú zobrazované v hlavnej časti administrátorského rozhrania. Každý prvok v navigačnom paneli má vlastnú ikonku a predstavuje rozdielnu časť rozhrania na serveri.

Hlavnú časť administrátorského rozhrania nazývame *panel*. Panel má všeobecnú štruktúru, ktorú rozdeľujeme na:

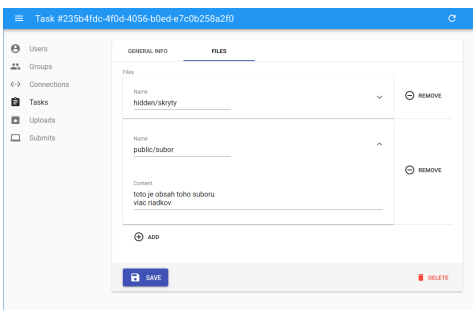
- akcie
- tabuľka entít
- stránkovanie

Akcie

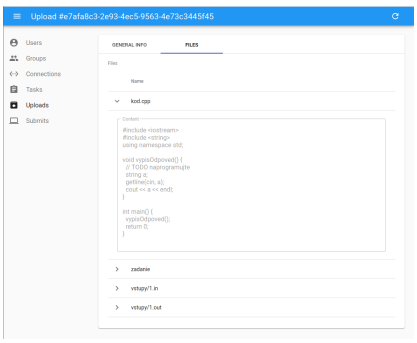
Akcie sa nachádzajú v hornej časti panela a štandardne je v knižnici React admin implemntovaná iba akcia *exportovať*, ktorá vytvorí CSV súbor so všetkými entitami zobrazenými v tabuľke. Ďalšie akcie sa však dajú pridať.Väčšina panelov má implementovanú aspoň akciu *vytvor*, ktorá ponúkne jednoduchý formulár na vyplnenie údajov entity, ktorý sa následne odošle na server, kde sa uloží do databázy. Okrem toho poskytuje knižnica možnosť vytvoriť si filtre. Filtre len pridávajú parametre, ktoré sa posielajú spolu s požiadavkou na rozhranie servera. Samotné filtrovanie sa vykonáva až na serveri.

Tabuľka entít

Tabuľka entít zobrazuje dáta, ktoré vracia rozhranie servera. Na názvy stĺpcov tabuľky sa dá kliknúť a slúžia na triedenie zobrazovaných výsledkov. Podobne ako pri filtrovaní, knižnica iba posiela parametre na server, ktorý zodpovedá za to, aby vrátené výsledky boli správne utriedené. Po kliknutí na riadok tabuľky sa zobrazí detailný popis danej entity. Knižnica umožňuje aj meniť údaje entít, záleží na tom, či chceme aby sa daná entita, prípadne nejaká jej časť dala zmeniť.



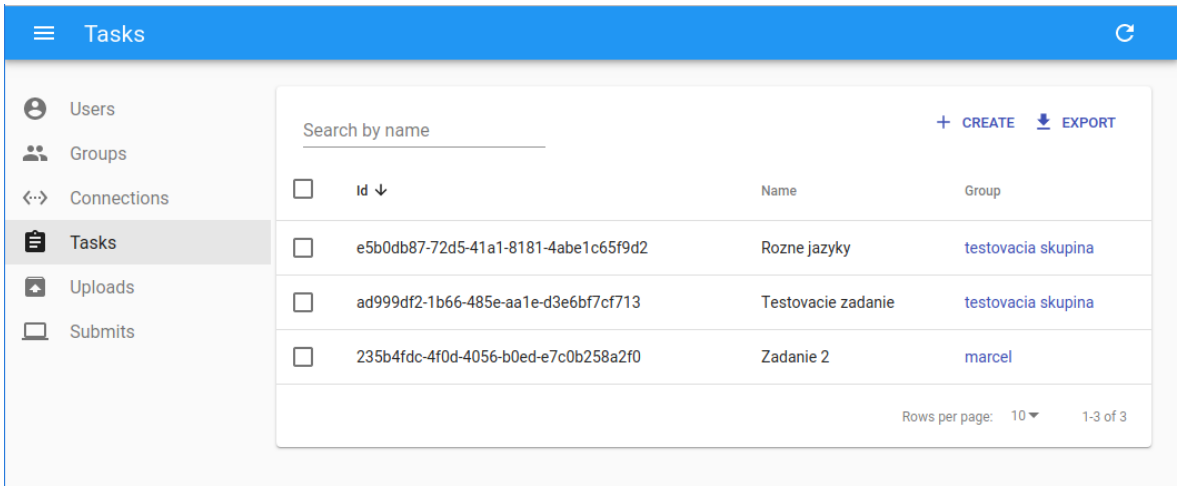
(a) Zmena zadania



(b) Detail nahratia zadania

Obr. 5.15: Detail entít

Tabuľka entít vie aj previazať vzťahy medzi entitami. Napríklad v prípade entity používateľov, zobrazené skupiny predstavujú klikateľný link na detail danej skupiny. Tiež si treba uvedomiť, že entita používateľa zo servera nedostane názov skupín, ale iba identifikátory skupín. Knižnica však dokáže pre každý identifikátor stiahnuť dáta o skupine a potom zobraziť meno danej skupiny.



Obr. 5.16: Tabuľka zadaní

Stránkovanie

V spodnej časti panela sa nachádza stránkovanie, ktoré sa využíva na to, aby v tabuľke nebolo naraz príliš veľa výsledkov. Stránkovanie je tiež užitočné kvôli šetreniu siete a rýchlosti odpovedania na požiadavky, pretože v odpovedi na požiadavku stačí posielať menej výsledkov. Počet entít v stránke je tiež modifikovateľný.

5.1 Server

Server, je rovnako ako klient, napísaný v jazyku TypeScript. Kód napísaný v tomto jazyku sa automaticky preloží na ekvivalentný kód jazyka JavaScript. Server potom tento kód spustí pomocou *Node.js*. Node je prostredie, ktoré slúži na spúštanie jazyka JavaScript mimo webového prehliadača. Okrem toho Node obsahuje viaceré knižnice, ktoré nie sú dostupné v prehliadači ako napríklad knižnica na prístup k súborovému systému alebo knižnica na spúštanie systémových volaní. Knižnica použitá na vytvorenie webového rozhrania servera sa nazýva *Express*. Express má viaceré rozšírenia, ktoré sú dostupné ako samostatné knižnice, ktoré napríklad komprimujú odpovede na webové požiadavky, prekladajú formát parametrov poslaných v požiadavke...

Server je zodpovedný za uchovávanie a menežovanie entít. Entity sa uchovávajú v databáze a priamo na súborovom systéme servera. V databáze sa ukladá všetko okrem súborov zadania a súborov vytvorených pri ukladaní zadania. Tieto súbory sú uložené

v špeciálnych priečinkoch, ktoré majú názov rovnaký ako identifikátor danej entity v databáze. To znamená, že ak server pozná identifikátor entity, vie si jednoducho načítať aj súbory patriace k danej entite. Použitá databáza je *PostgreSQL*, pretože poskytuje viaceré pokročilé funkcie.

Webové rozhranie servera umožňuje zadávať rôzne parametre, ktoré ovplyvňujú vrátené výsledky. Tieto parametre slúžia ako filtre, prípadne triedia výsledky podľa nejakého stĺpca. Tieto parametre sa posielajú databázovej vrstve, ktorá ich nastaví pri vytváraní databázovej požiadavky.

Spúštací skript

Okrem menežovania entít, je server zodpovedný aj za spúštanie kódu. Kód sa spúšta buď na vlastnom vstupe alebo na skrytých vstupoch zadania. To, ako sa má program spustiť, skompilovať a prípadne otestovať, hovorí kompilačný skript. Je to skrytý súbor, ktorý má názov *run_ script.json*. Tento súbor server načíta ešte predtým, ako kód klienta spustí. Server vďaka tomu vie tento skript pozmeniť a napríklad nastaviť, aby sa kód spúštal na vlastnom vstupe, ktorý rozhranie servera dostalo vo webovej požiadavke. Ak sa v kompilačnom skripte nenastaví vlastný vstup, tak program sa postupne spustí na všetkých skrytých vstupných súboroch a porovnáva ich výstup so správnym výstupom. Skryté súbory majú koncovku *.in* a výstupné *.out*.

V spúšťacom skripte vie používateľ nastaviť viaceré parametre testovania napríklad spôsob kompilácie a spúštania, časový limit...

```
{
  "compiler": "'g++ -o /usercode/a.out' ",
  "sources": "public/kod.cpp",
  "executable": "/usercode/a.out",
  "compilerName": "C++",
  "additionalArguments": ""
}
```

Obr. 5.17: Spúštací skript

5.2 Synchronizácia editora medzi klientami

Ak sa používateľ prihlási na viacerých počítačoch naraz, kód editora sa medzi nimi synchronizuje. Synchronizáciu zabezpečuje knižnica *yjs*, ktorá je nainštalovaná aj na klientovi a aj na serveri. Táto knižnica používa bezkonfliktné replikovateľné dátové typy a skrýva celú ich implementáciu. Knižnica vie synchronizovať veľa verejne dostupných editorov ako Monaco, ktorý používame. Knižnica vie zabezpečiť zdieľanie viacerými spôsobmi ako napríklad WebRTC alebo pomocou soketov cez webový server.

Knižnica však nie je až tak populárna a obsahovala viaceré chyby, ktoré sme museli opraviť. Komunita starajúca sa o túto knižnicu je o dosť menšia v porovnaní s ostatnými knižnicami použitými v bakalárskej práci.

5.3 Izolované spúštanie kódu na serveri

Kód spustený na serveri, nesmie narušiť ostatné bežiace programy. Klientský kód, môže obsahovať ľubovolné príkazy jazyka, ako napríklad systémové volania, ktoré sú pre ostatné programy a stav servera nebezpečné. Okrem bezpečnosti, je dôležité, aby kódy používateľov, ktoré sa práve spúštajú, boli nezávislé.

Myšlienka testovačov pre rôzne jazyky je tu už dlho, a problematika tejto témy je hodná celej bakalárskej práce. Rozhodli sme sa pre jednoduché riešenie, ktoré je škálovateľné pre ľubovolné jazyky. Riešenie je založené na knižnici *Compilebox*, ktorú sme mierne upravili. Základnou myšlienkou knižnice je, že kód sa spúšta v *dokeri*, ktorý zaručuje izolovanosť od ostatných procesov na serveri.

5.3.1 Doker

Doker je verejne prístupný projekt, ktorého cieľom je poskytnúť jednotné rozhranie pre izoláciu aplikácii do kontajnerov nezávislých od operačného systému. V dokeri vieme popísať, aké balíčky a knižnice do systému chceme nainštalovať, prípadne napísať nejaké iné systémové príkazy. Z takéhoto popisu vieme následne vytvoriť *Dokerový obraz*. Dokerový obraz predstavuje prostredie, v ktorom sú nainštalované všetky balíčky, ktoré sme uviedli v popise. Dôležité je, že balíčky existujú iba vo vnútri dokerového obrazu. Z dokerového obrazu sa dá vytvoriť inštancia, ktorá sa nazýva *dokerový kontajner*. V kontajneri sa dajú následne spúštať programy, ktoré majú prístup k balíčkom nainštalovaným v dokerovom obraze. Kontajnerov môže byť naraz pustených ľubovoľne veľa a sú navzájom izolované.

5.3.2 Spúštanie kódu

Po nainštalovaní dokera je nutné vytvoriť dokerový obraz, ktorý obsahuje všetky podporované jazyky. Ak na server príde požiadavka spustiť kód používateľa, vytvorí sa súbor, do ktorého sa nakopírujú všetky súbory potrebné na testovanie. Klient má prístup iba k verejným súborom, ale pri testovaní potrebujeme aj skryté súbory zadania. Pri spúštaní programu vieme, ktoré zadanie používateľ rieši a teda vieme, ktoré skryté súbory skopírovať. Názov súboru je unikátky identifikátor, ktorý zaručuje nezávislosť spúštania viacerých programov naraz. Keď sú všetky súbory pripravené, z dokerového obrazu sa vytvorí kontajner, v ktorom sa kód skompiluje a spustí. Výstup programu z

testovania je presmerovaný do výstupného súboru a prípadné chyby alebo varovania pri kompilácii, či behu programu sa zapisujú do samostatných súborov, ktoré kontajner vytvorí. Po dokončení behu programu kontajner vytvorí špeciálny súbor, ktorý označuje ukončenie behu programu. Server pravidelne kontroluje, či daný súbor existuje. Server úmyslne nečaká, kým sa spustený program v dokeri neskončí, pretože nič nezaručuje, že klientský program niekedy skončí. Vďaka takémuto riešeniu vieme implementovať aj časový limit programu, kde po prekročení daného limitu bežiaci program ukončíme. Po skončení behu programu vrátime obsah súborov serveru a celý priečinok, v ktorom boli potrebné súbory na spúštanie vymažeme.

Kapitola 6

Využiteľnost v praxi a podobné programy

6.1 Využiteľnost v praxi

Myšlienka kolaboratívnych editorov a všeobecne kolaboratívnych prostredí sa dá v praxi využiť vo viacerých oblastiach. Už v roku 2012 bola veľkosť latencie veľmi blízko teoretickému limitu, ktorý vieme dosiahnuť [1]. Stále sa však zlepšuje prístupnosť internetu ľuďom. V súčasnej dobe je bežné, že ľudia pracujúci za počítačom robia veci z pohodlia domova. Kolaboratívne programy umožnujú prepojenie týchto vzdialených používateľov a umožnujú im zdieľanie a upravovanie dokumentov.

Kolaboratívne editory sú populárne za účelom vzdialených pohovorov, kde pohovorujúci komunikuje s pohovorovaným cez internet alebo telefón a ten môže programovať do editora, pričom pohovorujúci vidí zmeny okamžite. Veľkou výhodou je, že pohovorovaný nemusí cestovať do kancelárie firmy, u ktorej vykonáva pohovor.

Ďalším využitím je párové programovanie. Striktná definícia párového programovania umožnuje písať naraz iba jednému, zatiaľ čo druhý je len pozorovateľ. Častokrát je však výhodné mať možnosť zasiahnuť do kódu aj ako pozorovateľ, prípadne programovať naraz.

Ďalším veľkým využitím, sú spoločné projekty ako také. *(Pod pojmom "projekt"máme na mysli ľubovolnú prácu alebo zadanie, ktoré vykonáva viacero ľudí)*. Môže ísť o projekt vo firme, školské zadanie... Častokrát sa tieto projekty dajú rozdeliť na zmysluplné časti, ktoré vedia byť spravené nezávisle od seba.

Posledným využitím a zároveň aj cieľom tejto práce bolo vytvoriť učebné prostredie, ktoré by sa na univerzite dalo používať na informatických predmetoch. Učiteľ by zadal úlohy, ktoré by študenti vedeli riešiť, pričom by mali možnosť pracovať na zadaní súčasne.

6.2 Porovnanie s existujúcimi programami

Ako sme už spomenuli v kapitole 2, existuje viacero programov, ktoré podporujú kolaboratívne upravovanie, pričom stále vznikajú nové. Sústreďme sa ale iba na porovnanie s programami, ktoré podporujú kolaboratívne úpravy **textového** dokumentu. Tieto programy sa dajú rozdeliť do viacerých kategorií, ktoré vyjadrujú odlišnosť oproti návrhu tejto bakalárskej práce. Programy, ktoré tu budú spomenuté, sa odlišujú buď v použitých technológiách alebo tým, že nie sú zamerané na programovanie zdrojových kódov(a teda nepodporujú ani spúštanie a testovanie), prípadne podporujú iba nejaké jazyky.

6.2.1 Editory využívajúce technológie dostupné iba na niektorých platformách

Do tejto kategórie patria programy vytvorené pre operačné prostredie *macOs*. Príkladom takéhoto programu je *SubEthaEdit*, ktorý začal revolúciu kolaboratívneho programovania. V súčasnosti je editor dostupný zadarmo a je open source. Ďalším rozdielom oproti návrhu v bakalárskej práci je nemožnosť daný kód zbiehať.

6.2.2 Editory používajúce inú technológiu konkurentých úprav

Na spracovávanie konkurentých úprav poznáme dve dobre preskúmané technológie (*OT* alebo *CRDT*). Väčšina produktov používa *CRDT*, lebo je to výrazne ľahšie na implementáciu a udržiavanie. Medzi programy využívajúce OT patria editory *SubEthaEdit, ACE, Gobby, MoonEdit*... Mnohé z nich prestali byť ďalej vyvíjané alebo sa o ne stará open source komunita.

6.2.3 Online dostupné textové editory

Ďalšou kategóriou sú online editory, teda webové aplikácie. Niektoré su veľmi populárne ale poskytujú iba čiastočnú funkcionalitu. Napríklad *Ideone* je editor, v ktorom sa dá zadarmo spúštať kód vo viac ako 50 jazykoch. Neumožňuje však kolaboratívne editovanie a testovanie na skrytých vstupoch. Jedným z najpoužívanejších voľne dostupných editorov je *CodeShare*, ktorý umožnuje kolaboratívne písanie, podporuje zvýrazňovanie syntaxe mnohých jazykov a obsahuje ďalšie dopĺnujúce prvky, ako videokamera pre vytvorenie editora spolu s videom.

Ako posledný si predstavme *CodeSandbox*, ktorý podporuje iba písanie v jazyku JavaScript, je však zaujímavý tým, že je open source a používa veľmi pokročilý editor VSCode, čo je ďalší open source produkt. *CodeSandbox* umožnuje aj kolaboratívne

upravovanie. Je však zameraný iba na JavaScript a je používaný hlavne na rýchle skúšanie webových frameworkov alebo krátkych útržkov javascriptových kódov. Výhodou je, že používateľ nemusí nič inštalovať, lebo *CodeSandbox* má všetko potrebné nainštalované na serveri. Okrem krátkych skúšaní projektov, umožnuje aj pohodlnú integráciu s gitom, pridávanie ďalších knižníc...

6.2.4 Platené programy

Poslednou veľkou skupinou sú platené programy, ktoré poskytujú viac funkcionalít, ale ich použitie je platené. Príkladmi týchto editorov sú *CodeBunk* a *CoderPad*. Mnohé z nich poskytujú aj spúštanie kódu, prípadne videokameru.

Záver

Podarilo sa nám implementovať celý návrh programu, ktorý je pripravený uľahčiť výučbu programovania ako aj vytváranie jednoduchých zadaní pre študentov. Implementácii predchádzalo študijné obdobie zaoberajúce sa fungovaním zdieľateľnosti editorov a izolovaným spúšťaním kódu. V samotnej implementácii sme použili viaceré moderné knižnice, ktoré nám uľahčili vývoj programu. Súčasný stav aplikácie je použiteľný, ale len pre testovacie účely. Aplikácia nijako nerieši bezpečnosť a študenti môžu veľmi ľahko získať prístup k dátam, ktoré by mali byť dostupné iba pre administrátora. Okrem toho prostredie nezabezpečuje spôsob testovania zdrojových kódov dokonale. Najlepšie by bolo, ak by sa kód testoval na testovači v skutočnom virtuálnom prostredí, čo doker neposkytuje.

Zdrojový kód aplikácie je spravovaný prostredníctvom služby GitHub, ktorá umožňuje spoluprácu pri vytváraní voľne šíriteľných programov. Pri implementácii boli dodržiavané štandardy programovania v jazyku TypeScript, SQL a knižnice React. Okrem toho je zdrojový kód písaný v anglickom jazyku, takže na knižnici môžu pracovať aj študenti zo zahraničia. Knižnica je teda pripravená na prípadný budúci rozvoj.

Literatúra

[1] Latency: The new web performance bottleneck, July 2012. https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck.

[2] Building conclave: a decentralized, real time, collaborative text editor, January 2018. https://hackernoon.com/building-conclave-a-decentralized-real-time-collaborative-text-editor-a6ab43

[3] X. Autor1 and Y. Autor2. *Názov knihy.* Vydavateľstvo, 1900.

[4] X. Autor1 and Y. Autor2. Názov článku (väčšinou z konferencie). In *Názov zborníka (väčšinou názov konferencie spolu s ročníkom)*, pages 1–100, 1900.

[5] X. Autor1 and Y. Autor2. Názov článku z časopisu. *Názov časopisu, ktorý článok uverejnil*, 4(3):1–100, 1900.

[6] X. Autor1 and Y. Autor2. Názov technickej správy. Technical Report TR123/1999, Inštitút vydávajúci správu, June 1999.

[7] Abdessamad Imine, Michaël Rusinowitch, Gérald Oster, and Pascal Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theor. Comput. Sci.*, 351(2):167–183, February 2006.

[8] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *Nie príliš stručný úvod do systému LaTeX2e.* 2002. Preklad Ján Buša ml. a st.

[9] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, and Mihai Letia. A commutative replicated data type for cooperative editing. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 395–403, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*, SSS'11, pages 386–400, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] Univerzita Komenského v Bratislave. Vnútorný predpis č. 12/2013, smernica rektora Univerzity Komenského v Bratislave o základných náležitostiach záverečných prác, rigoróznych prác a habilitačných prác, kontrole ich originality, uchovávaní a sprístupňovaní na Univerzite Komenského v Bratislave, 2013. https://uniba.sk/fileadmin/ruk/legislativa/2013/Vp_2013_12.pdf.

Príloha

V prílohe sa nachádza komplentný zdrojový kód, ktorý je dostupný aj cez službu GitHub. Projekt je rozdelený do samostatných priečinkov pre klienta a pre server. Oba tieto priečinky obsahujú popisný súbor, v ktorom je napísané ako projekt spustiť.