Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Inteligentný monitorovací systém práce študentov v Linuxovom termináli

Bakalárska práca

2018
Adrián Matejov

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Inteligentný monitorovací systém práce študentov v Linuxovom termináli

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	RNDr. Marek Nagy, PhD.

Bratislava, 2018
Adrián Matejov

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Adrián Matejov
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: Inteligentný monitorovací systém práce študentov v Linuxovom termináli
Intelligent System for Monitoring Students who Work in the Linux Terminal

Anotácia: Cieľom je vytvoriť webovú aplikáciu pomocou nástrojov node.js, HTML5, JavaScript,... Aplikácia bude real-time prijímať údaje o riešení úloh študentov, ktoré realizujú pomocou terminálovej aplikácie GTA. Okrem samotného monitorovania bude aplikácia odhadovať kritické situácie a informovať administrátora. Napríklad identifikovanie stagnujúcich študentov pri riešení problému, automatické sledovanie a podchytenie rôznych spôsobov riešenia, ...

Vedúci: RNDr. Marek Nagy, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 17.10.2017

Dátum schválenia: 26.10.2017 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcel by som sa poďakovať môjmu školiteľovi RNDr. Marekovi Nagyovi, PhD. za poskytnuté konzultácie, cenné rady a možnosť testovať systém na cvičeniach. Vďaka patrí Marekovi Šuppovi za mnoho nápadov, ochotu diskutovať o vylepšeniach systému a pomoc s odosielaním dát. Ďakujem aj mojej rodine a kamarátom, ktorí ma podporovali počas štúdia.

Abstrakt

V tejto práci predstavujeme administratívny a monitorovací systém, ktorý slúži na monitorovanie priebehu cvičenia, kde študenti pracujú v Linuxovom termináli. Systém sa snaží automaticky detegovať neaktivitu alebo zaseknutie študentov pri konkrétnych úlohách a okamžite upozorňovať vyučujúceho. Takto dostáva učiteľ oveľa lepší prehľad o napredovaní študentov. Po cvičení ponúka rozhranie na automatické ohodnotenie riešení a detekciu rôznych spôsobov riešenia úloh pomocou metódy K-means.

Kľúčové slová: monitorovanie, Node.js, Linux, K-means

Abstract

This thesis introduces administrative and monitoring system for monitoring the course of seminar, where students work in the Linux terminal. The system automatically detects inactivity of students or students being stuck on particular task and immediately notifies the teacher about them. This way the teacher has much better overview of progresses of students. After the seminar, the system provides interface for automatic evaluation of assignments and detection of different ways of solving tasks using K-means method.

Keywords: monitoring, Node.js, Linux, K-means

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Monitorovacie systémy nám pomáhajú zbierať dáta, upozorňovať používateľov na rôzne anomálie, vizualizovať a robiť prehľad historických dát. Dáta, ktoré takto získame, nám vedia poskytnúť veľmi dôležitú spätnú väzbu o fungovaní systému, ktorý monitorujeme.

Táto práca vznikla na podnet vyučujúcich kurzu Linux pre používateľov, ktorým chýbal takýto prehľad o cvičeniach. V práci sa zameriavame na vývoj aplikácie, ktorá monitoruje priebeh cvičení, na ktorých študenti riešia úlohy v Linuxovom termináli. Aplikácia v reálnom čase upozorňuje vyučujúcich na rôzne udalosti ako napríklad neaktivitu študentov alebo zaseknutia sa v úlohe. Po cvičení poskytuje rozhranie na vyhodnotenie týchto aktivít. Dôležitou súčasťou nášho systému je aj podchytenie rôznych spôsobov riešení, ktorými sa študentom podarilo vyriešiť zadania.

Práca sa skladá z niekoľkých častí. V kapitole 1 uvedieme základné informácie o monitorovacích systémoch, priblížime fungovanie kurzu Linux pre používateľov a predstavíme požiadavky na našu aplikáciu.

Kapitola 2 detailne popisuje návrh databázových tabuliek, tok dát a formát dát, ktoré zbierame od študentov. Taktiež zahŕňa informácie o programovacom jazyku, knižniciach a frameworkoch, ktoré sme použili pri vývoji.

V kapitole 3 sa venujeme implementačným detailom aplikácie. V tejto kapitole taktiež vysvetlíme fungovanie metódy strojového učenia *K-means*, ktorú využívame v práci.

Kapitola 4 prezentuje výsledky z testovania aplikácie počas semestra. Uvádzame v nej aj zhodnotenie aplikácie vyučujúcimi, ktorý ju využívali v rámci cvičení.

Kapitola 1

Špecifikácia

Táto kapitola obsahuje základné informácie o monitorovacích systémoch a textových adventúrach, kde všade sme sa s nimi mohli stretnúť, predstavuje základnú motiváciu pre našu bakalársku prácu a popisuje špecifikáciu našej aplikácie.

1.1 Monitorovacie systémy

Monitorovacie systémy sú neoddeliteľnou súčasťou pri vývoji a spravovaní rôznych komponentov hardvéru, softvéru, mechanických častí a podobne. Dávajú nám dôležité informácie o ich stave, vedia nás včas upozorňovať o potenciálnych hrozbách, zbierajú dáta, ktoré vieme vyhodnocovať a predísť podobným prípadom v budúcnosti. Priblížme si niektoré typy monitorovacích systémov.

Sieťové sledujú dostupnosť webových služieb, mailových serverov, využitie diskového priestoru, oneskorenie odpovede alebo možný DDoS útok. Následne vedia informovať administrátora o týchto skutočnostiach. Príkladom takéhoto systému je *Zabbix*. [1]

Softvérové zbierajú informácie o vyťažovaní procesorov, RAM pamäte, celkovom stave aplikácií. V unixových systémoch je na vizualizáciu týchto informácií využívaný program *top*, respektíve *htop*.

Hardvérové sú využívané napríklad na sledovanie teploty zariadenia, funkčnosti rôznych mechanických častí, tlaku v nádržiach s kvapalinami atď.

Na monitorovanie *počasia* sú taktiež využívané rôznorodé senzory a satelity. Pomocou nich vieme získavať cenné dáta o aktuálnom stave počasia takmer hocikde na zemi, ktoré sú zužitkované na predpovede alebo varovanie obyvateľstva pred blížiacou sa prírodnou katastrofou.

Všetky tieto systémy majú dôležité spoločné črty. Vedia zbierať dáta, spracovávať ich, vyhodnocovať, vizualizovať a informovať administrátorov a používateľov o neštandardnom správaní. My sa v tejto práci budeme venovať taktiež monitorovaciemu

systému, ktorý sleduje aktivitu študentov na cvičení.

1.2 Textové adventúry

Textové adventúry sú jednými z najstarších typov počítačových hier, ktoré spadajú pod dobrodružný žáner. Hráč využíva na interakciu v hre spravidla iba klávesnicu a hra následne odpovedá textovým výstupom. Vstup je väčšinou zadávaný používateľom vo forme krátkych slovných spojení typu „vziať kľúč“ alebo „choď vpred“, ktoré sú spracovávané textovým parserom. Prvé parsery rozumeli iba dvojslovným spojeniam typu sloveso + podstatné meno. Neskôr boli vyvinuté také, ktoré už rozumeli celým vetám.

Napriek neprítomnosti grafického prostredia môžu tieto typy hier popisovať prostredie, v ktorom sa hráč pohybuje, čo rozvíja predstavivosť. Taktiež môže vytvárať nelogické prostredie, kde z bodu A sa dá dostať do bodu B, ale z bodu B sa už reverzným pohybom nedostaneme do bodu A. [11]

1.3 Motivácia

1.3.1 go-term-adventures

Textovými adventúrami sú aj takzvané *go-term-adventures* (ďalej GTA) (`https://github.com/NaiveNeuron/go-term-adventures`) [15], ktoré vznikli na Fakulte matematiky, fyziky a informatiky Univerzity Komenského ako projekt pre kurz Linux pre používateľov. Hlavným motívom pre tvorbu GTA bolo zefektívnenie cvičení. Ako už samotný názov napovedá, GTA aplikácia je naprogramovaná v programovacom jazyku Go.

Študentov cvičením sprevádza vopred vygenerovaný samorozbaľovací skript. Cvičenie je rozdelené do takzvaných *levelov*. Každý level začína vypísaním zadania priamo do terminálu. Následne je používateľ vyzvaný napísať príkaz v termináli. Ak zadaný príkaz vráti správny výstup, skript pošle používateľa do ďalšieho levelu. Vďaka tomu študent nemusí prepínať medzi zadaním a terminálom, a taktiež má priamo overenú správnosť výstupu príkazu. Niektoré levely majú viac alternatív, kde každá má jemne pozmenené zadanie. Študenti riešia iba jednu alternatívu každého z levelov.

Skript počas práce študenta zachytáva všetky príkazy a zapisuje ich do textového dokumentu spoločne s časom vykonania. Ten po cvičení odošlú a prednášajúci ich vie obodovať.

1.3.2 Kurz Linux pre používateľov

Kurz Linux pre používateľov je úvodom do operačného systému Linux na používateľskej úrovni. Študenti sa učia pracovať v termináli so základnými príkazmi. Vytvárajú priečinky, vyhľadávajú a prepisujú určité úseky v texte, počítajú rôzne štatistické údaje z dát a podobne.

Do akademického roku 2015/2016 študenti na cvičeniach vytvárali príkazy v termináli, a keď si boli istí, že fungujú, tak ich skopírovali a poslali na obodovanie. Nemali tak informáciu o tom, či ich riešenie je naozaj správne. Zároveň učiteľ nemal prehľad o tom, akým postupom sa študenti dopracovali k riešeniu. Neskôr, od roku 2016/2017 vznikol projekt go-term-adventures (viď sekciu 1.3.1), vďaka ktorému sa cvičenia stali viac interaktívnejšími a aj práca na nich sa zefektívnila.

V akademickom roku 2016/2017 si kurz Linux pre používateľov na FMFI UK zapísalo 90 študentov. Na cvičeniach sa väčšinou snažilo pomáhať 6 cvičiacich. Študenti riešili úlohy a často sa stávalo, že potrebovali pomôcť a dlhšiu dobu sa neprihlásili. Pri tak veľkom počte študentov je ťažké zisťovať, kto sa zasekol na danej úlohe. Taktiež sa vyskytlo veľa problémov s príliš podobnými riešeniami a nemožnosť riešiť podozrenie z opisovania priamo na cvičení komplikovalo hodnotenie.

Riešenia úloh študenti posielali ako textový súbor so zoznamom príkazov do systému Moodle a hodnotiaci si ich vždy musel sťahovať, aby videl obsah. Človek je tvor zábudlivý, čo sa potvrdilo aj na tomto kurze. Mnohokrát sa stalo, že študent zabudol odovzdať tento textový súbor, a tým pádom mu hodnotiaci nemohol prideliť body za dané cvičenie, čo prinieslo mnoho problémov nielen študentovi, ale aj prednášajúcemu.

Pre naše potreby neexistoval žiadny monitorovací systém, ktorý by spracovával dáta nášho typu a zároveň ich vizualizoval v reálnom čase. Preto sme sa rozhodli naprogramovať aplikáciu, ktorá bude monitorovať priebeh cvičenia, a vyučujúcim tak poskytne prehľad nielen o riešeniach študentov ale aj administrátorské webové rozhranie na správu cvičení.

1.4 Požiadavky na aplikáciu

V nasledujúcich bodoch zhrnieme požiadavky na našu aplikáciu:

- Zbieranie dát z GTA aplikácie od všetkých študentov
- Ukladanie dát do databázy
- Zobrazovanie priebehu cvičenia všetkým klientom (vyučujúcim) v reálnom čase
- Prehľad historických cvičení
- Automatické obodovanie jednotlivých riešení a exportovanie bodov do csv súboru

- Detekcia neaktivity študentov
- Detekcia rôznych spôsobov riešenia úloh

Nižšie sa pozrieme na naše požiadavky podrobnejšie.

1.4.1 Zbieranie dát z GTA aplikácie od všetkých študentov

Aby sme mohli monitorovať aktuálny stav študentov, potrebujeme mať záznam o všetkých akciách, ktoré študenti vykonali v termináli počas cvičenia.

Každá z požiadaviek bude obsahovať základné informácie o počítači, na ktorom študent pracuje spolu s dátumom a časom odoslania, typom požiadavky a číslom cvičenia. Informáciu o IP adrese chceme využívať na to, aby študenti boli nútení riešiť cvičenie v škole. Študenti sa prihlasujú na počítače pod svojim univerzitným kontom, ktoré je unikátne. Tým pádom vieme jednoznačne priradiť študenta k počítaču, čo nám pomôže pri vizualizácii miestnosti.

1.4.2 Zobrazovanie priebehu cvičenia všetkým klientom (vyučujúcim) v reálnom čase

Cvičiacim chceme priniesť prehľad o aktuálnom stave všetkých študentov riešiacich cvičenie. Preto je potrebná pekná vizualizácia v reálnom čase. Nechceme, aby si museli inštalovať nejaký softvér, preto bude najlepšie využiť webový prehliadač, ktorý je v dnešnej dobe dostupný aj na tabletoch a smartfónoch. Zobrazovať chceme hlavne jednoznačný identifikátor študentov a počítačov, na ktorých pracujú, level, v ktorom sa aktuálne nachádzajú, čas od posledného príkazu, neaktivitu po dlhšej dobe, a taktiež počet pokusov pri prechádzaní aktuálneho levelu. Bude sa dať pozrieť aj detail o študentovi, kde zobrazíme čas začiatku, konca a všetky príkazy, ktoré napísal na cvičení.

Zároveň pre lepšiu orientáciu vyučujúcich v miestnosti ponúkneme možnosť preusporiadania počítačov v tejto vizualizácii, ktorú si systém zapamätá a na ďalšom cvičení bude schopný zrekonštruovať toto rozostavenie počítačov v učebni.

1.4.3 Prehľad historických cvičení

Študenti po skončení kurzu hodnotili tento kurz v študentskej ankete. Veľa z nich sa vyjadrilo, že niektoré úlohy boli náročné a namiesto jednej vyučovacej hodiny strávili na cvičení dve až tri. Je dôležité, aby si tvorcovia jednotlivých úloh na cvičenie vedeli spätne pozrieť ich náročnosť a poprípade v budúcnosti upraviť zadanie. Zaujímavé dáta, ktoré chceme o každom študentovi zobraziť, sú čas strávený riešením cvičenia a počet príkazov. Ďalej sú to priemery a mediány časov riešenia všetkých študentov a

počtu príkazov. O každej úlohe chceme vedieť počet úspešných a neúspešných pokusov o riešenie.

1.4.4 Automatické obodovanie jednotlivých riešení a exportovanie bodov do csv súboru

Kurz prebieha v e-learningovom systéme Moodle. Každý študent si tam vie prezerať svoje body za cvičenia, testy, nájde tam prednášky a iné materiály. Systém Moodle akceptuje aj formát *csv*, v ktorom prednášajúci nahrá súbor hodnotenia a body budú automaticky pridelené študentom. Preto chceme, aby náš systém vedel poskytnúť rozhranie pre automatické obodovanie jednotlivých úloh a ich následný export v csv formáte.

1.4.5 Detekcia neaktivity študentov

Študenti sa niekedy zaseknú na úlohe a nevedia ako riešiť daný problém, poprípade majú problém pochopiť zadanie. Náš systém bude detegovať neaktívnych študentov, ktorí nevykonali žiadnu akciu po dobu n minút. Systém následne upozorní vyučujúcich o tejto skutočnosti a tí môžu pomôcť študentom. Taktiež bude upozorňovať vyučujúcich, pokiaľ sa študentovi nepodarí prejsť level po určitom počte príkazov.

1.4.6 Detekcia rôznych spôsobov riešenia úloh

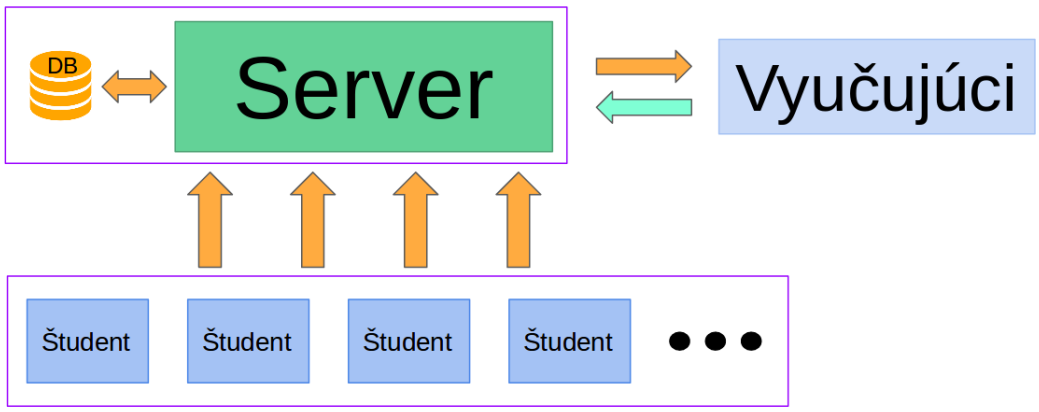
Veľa z úloh, ktoré sa vyskytujú na cvičeniach má viacero spôsobov riešenia. Vo väčšine prípadov aj študenti nerozmýšľajú úplne rovnako pri riešení. Preto chceme vyučujúcemu poskytnúť cennú spätnú väzbu o týchto spôsoboch. Získa tak prehľad o tom, či študenti porozumeli prednáške a v budúcnosti sa tak môže zamerať na zlepšenie výkladu učiva, poprípade sa viac sústrediť na to, čo študentom nešlo. Taktiež tento prehľad môže byť použitý pri podozrení na plagiátorstvo.

Kapitola 2

Návrh aplikácie

V tejto kapitole uvedieme informácie o architektúre našej aplikácie, návrhu databázy a význam jej tabuliek, spôsob, akým budeme prenášať údaje od študentov, zadefinujeme akcie, ktoré vyvolávajú tento prenos a popíšeme fremework Node.js spolu s modulmi, ktoré sme sa rozhodli použiť.

2.1 Architektúra toku dát



Obr. 2.1: Architektúra toku dát.

Obrázok 2.1 ilustruje tok dát. Každý študent v učebni pracuje na počítači v Linuxovom termináli. V ňom beží aplikácia GTA, ktorá odosiela dáta priamo na server šifrované pomocou TLS. Ten ich spracuje, patrične uloží v databáze a okamžite pošle vyučujúcim, kde sa zobrazia vo webovom rozhraní.

Vyučujúci môžu napríklad vytvárať nové inštancie cvičenia, hodnotiť riešenia študentov. Dáta v tomto prípade posielame POST požiadavkami na server. Spojenie medzi klientom (vyučujúcim) a serverom je taktiež šifrované.

2.2 Návrh databázy

Tabuľky sme navrhli do nasledujúcej štruktúry (viď 2.2). Väčšina tabuliek sa odkazuje na cudzí kľúč *exercise_id* z tabuľky *Exercise*. Tá reprezentuje konkrétnu inštanciu cvičenia. Má nastavený svoj špecifický identifikátor, meno, posledný level, čas začiatku a konca a stav, v akom sa nachádza. Možné stavy cvičenia sú *scheduled* (naplánované), *active* (prebiehajúce), *done* (ukončené).

V tabuľke *Post* sa ukladajú dáta, ktoré posiela GTA aplikácia. Bližšie informácie o týchto dátach sú zhrnuté v sekcii 2.3.

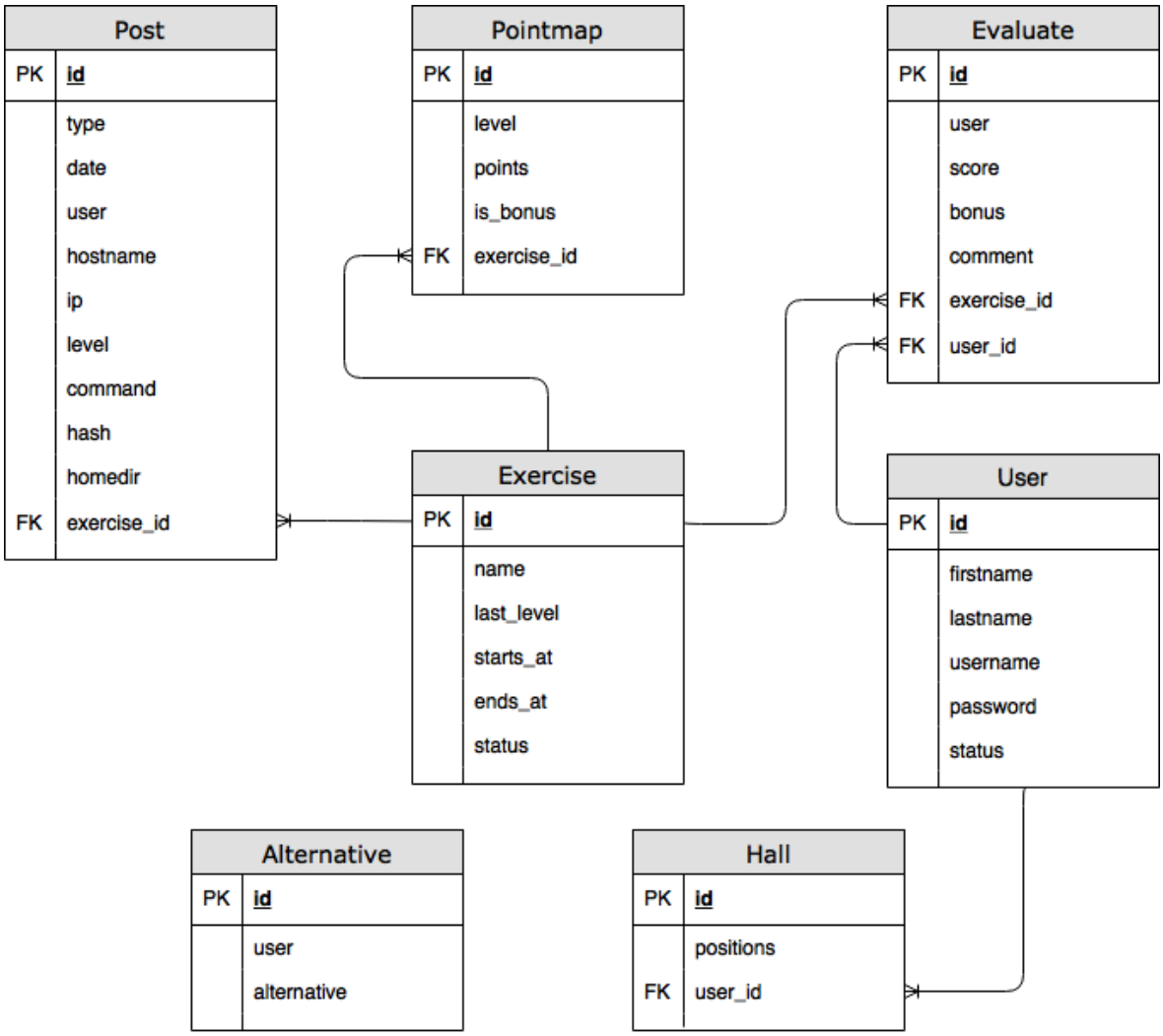
Do tabuľky *Pointmap* si zapíšeme koľko bodov bude pridelených študentovi za jednotlivé prejdené levely pri hodnotení. Položka *is_bonus* nám hovorí, či tieto body budú pridelené do súčtu k bonusovým alebo nebonusovým. V aplikácii GTA sa často vyskytuje aj viacero alternatív jedného levelu. Preto položka *level* môže obsahovať regulárny výraz, ktorý bude vzorom pre viacero levelov.

Všetkým vyučujúcim sa uloží vytvorený účet do tabuľky *User*. Heslo sa ukladá do stĺpca *password*. Aby sme skryli heslo pred útočníkom, je zahašované funkciou *bcrypt* pred uložením do databázy.

V požiadavkách na systém je aj možnosť obodovať jednotlivé riešenia študentov. Tieto body sa ukladajú do tabuľky *Evaluate*. Pri hodnotení budeme rozlišovať body za štandardné a body za bonusové úlohy. K hodnoteniu môže vyučujúci pridať aj komentár, aby študent získal spätnú väzbu k svojmu riešeniu. V tabuľke sa nachádzajú dva cudzie kľúče. Prvý, *exercise_id*, odkazuje na cvičenie, v ktorom je študent hodnotený. Druhý, *user_id*, odkazuje na vyučujúceho, ktorý hodnotil študenta.

Každý vyučujúci má možnosť si rozmiestniť študentov na plátno. Systém si pamätá toto rozmiestnenie na základe takzvaného *hostname* počítača, na ktorom študent pracuje. Toto rozmiestnenie uložíme ako JSON objekt skonvertovaný do textu v stĺpci *positions* a priradíme konkrétnemu vyučujúcemu.

Tabuľka *Alternative* slúži na nastavenie alternatívneho mena pre študenta. Keďže niektorí študenti pracujú na vlastných počítačoch, ich používateľské meno nekorešponduje s menom v systéme Moodle a pri importovaní bodov z csv súboru by nastala chyba.



Obr. 2.2: Návrh databázových tabuliek a vzťahov medzi nimi.

2.3 Zbieranie dát z GTA aplikácie

Dáta budú na server posielané požiadavkami typu POST v široko používanom JSON (JavaScript Object Notation) formáte, ktorý ukladá dáta ako *kľúč: hodnota*. JSON je nezávislý na použití programovacieho jazyka. Pôvodne bol odvodený z JavaScriptu. Pracuje sa s ním veľmi jednoducho a dáta sú pre človeka ľahko čitateľné.

Každá požiadavka bude obsahovať nasledujúce základné informácie:

```
{
    "type": "typ poziadavky",
    "user": "$USER",
    "hostname": "‘hostname‘",
    "ip": "ip pocitaca",
    "exercise_number": "ID cvicenia",
    "date": "datum a cas"
}
```

Špecifikujme ďalej rôzne typy POST požiadaviek. Tie definujú všetky akcie, ktoré môžu nastať pri riešení úloh.

Typ *start*

Keď študent spustí GTA skript, systém si poznačí študenta, ktorý začína riešiť cvičenie.

```
{
    "type": "start",
    "user": "$USER",
    "hostname": "‘hostname‘",
    "ip": "ip pocitaca",
    "exercise_number": "ID cvicenia",
    "date": "datum a cas"
}
```

Typ *command*

Keď študent napíše v termináli príkaz a stlačí enter, odošleme informáciu o príkaze. Položka *level* označuje level, v ktorom sa študent nachádza. Položka *command* označuje odoslaný príkaz.

```
{
    "type": "command",
```

```
    "user": "$USER",
    "hostname": "‘hostname‘",
    "ip": "ip pocitaca",
    "exercise_number": "ID cvicenia",
    "date": "datum a cas",
    "level": "ID levelu",
    "command": "prikaz z terminalu",
}
```

Typ *passed*

Keď študent prejde level, namiesto typu *command* odošleme *passed*, keďže chceme mať záznam o tom, akým príkazom sa študentovi podarilo prejsť daný level.

```
{
    "type": "passed",
    "user": "$USER",
    "hostname": "‘hostname‘",
    "ip": "ip pocitaca",
    "exercise_number": "ID cvicenia",
    "date": "datum a cas",
    "level": "ID levelu",
    "command": "prikaz z terminalu",
    "hash": "hash uspesnosti cvicenia",
    "homedir": "$HOME"
}
```

Typ *exit*

Keď študent ukončí cvičenie napísaním *exit* v termináli, systém si poznačí, že daný študent skončil a môže zaradiť jeho riešenie na ohodnotenie.

```
{
    "type": "exit",
    "user": "$USER",
    "hostname": "‘hostname‘",
    "ip": "ip pocitaca",
    "exercise_number": "ID cvicenia",
    "date": "datum a cas",
    "hash": "hash uspesnosti cvicenia",
```

```
    "homedir": "$HOME"
}
```

Typ *help*

Keď študent potrebuje pomôcť s úlohou, odošle príkaz, ktorý upozorní vyučujúcich. Záznam sa uloží do databázy a vyučujúci tak získa prehľad o počte žiadostí o pomoc v jednotlivých leveloch.

```
{
    "type": "help",
    "user": "$USER",
    "hostname": "‘hostname‘",
    "ip": "ip pocitaca",
    "exercise_number": "ID cvicenia",
    "date": "datum a cas",
    "level": "ID levelu"
}
```

Typ *ack*

Po úspešnej pomoci študentovi odošle vyučujúci alebo študent túto informáciu do našej aplikácie a vytvorí sa záznam v databáze.

```
{
    "type": "ack",
    "user": "$USER",
    "hostname": "‘hostname‘",
    "ip": "ip pocitaca",
    "exercise_number": "ID cvicenia",
    "date": "datum a cas",
    "level": "ID levelu"
}
```

2.4 Node.js

Node.js je multiplatformový open-source softvér, ktorý spúšťa JavaScriptový kód na strane servera. Je primárne určený na programovanie webových serverov. Historicky sa JavaScript používal ako skriptovací jazyk na strane klienta, kde tento skript bol

vložený v HTML stránke a prehliadač ho vedel stiahnuť a spustiť. Keďže Node.js ponúka vývojárom spúšťanie skriptov na strane servera, otvára sa možnosť na dynamické generovanie stránok predtým, ako sú odoslané do prehliadača. Ďalej utilizuje takzvanú „JavaScript všade“ paradigmu, ktorá hovorí o tom, že kód webových aplikácií na strane servera a na strane klienta nepoužíva rozdielne programovacie jazyky. [8]

Prvýkrát bola táto technológia predstavená na konferencii European JSConf v roku 2009. Jej autor, Ryan Dahl, kritizoval, že majoritná väčšina programovacích jazykov na strane servera zbytočne čaká na odpoveď vstupno-výstupnej operácie, alebo výsledku z databázy. Preto cieľom Node.js je poskytnúť neblokujúci a udalosťami riadený model. Dovoľuje teda zvýšiť priepustnosť aplikácie, ktorá by inak bola omnoho nižšia.

Node.js sa silno spolieha na využívanie modulov pri vývoji aplikácií. Združuje ich takzvaný Node Package Manager (NPM). Pri implementácii našej aplikácie sa opierame o niekoľko z nich. Nižšie si uvedieme podrobnejšie informácie o týchto moduloch. [5]

2.4.1 Express.js

Express.js je jeden z najpoužívanejších frameworkov pre Node.js. Jeho flexibilnosť a minimalizmus ponúkajú pohodlný a rýchly vývoj webových aplikácií. Veľa vylepšení je dostupných v podobe pluginov, ktoré si môžeme nainštalovať z NPM databázy. My využijeme architektúru *model-view-controller* (MVC). Najskôr príde požiadavka na server, ktorú obsluhuje *controller*. Pri obsluhovaní interaguje s databázovým *modelom*. Následne sa zavolá *view*, ktorý vyrenderuje stránku a tá sa odošle klientovi do prehliadača. Na renderovanie stránky využijeme view engine *Pug*, ktorý má plnú podporu pre Express.js aplikácie.

2.4.2 Sequelize

Aby sme vedeli ukladať a manipulovať dáta v databáze, potrebujeme modul *mysql2*. Avšak, všetky dotazy do databázy treba písať v jazyku sql, čo môže vytvoriť nemalé množstvo chýb a zneprehľadniť kód.

Objektovo relačné zobrazenie (ORM) je technika softvérového inžinierstva, ktorá umožňuje automaticky konvertovať dáta medzi databázou a objektovo orientovaným programovacím jazykom. [4]

Sequelize je ORM modul pre Node.js aplikácie. Podporuje relačné databázy *MySQL*, *MariaDB*, *SQLite*, *PostgreSQL* a *MsSQL*. Vďaka prehľadnej dokumentácii [3], dobrej funkcionalite a veľkej komunite sme sa rozhodli použiť Sequelize v našej aplikácii.

2.4.3 WebSocket a Socket.io

WebSocket [6] je komunikačný protokol, ktorý poskytuje full-duplex asynchrónnu komunikáciu cez jediné TCP spojenie. Využíva *ws* (nezabezpečený) alebo *wss* (zabezpečený) protokol. Nie je limitovaný ako napríklad AJAX, kde klient musí najskôr vytvoriť požiadavku na server. Server aj klient si môžu posielať správy jeden druhému bez obmedzenia. Problém s WebSocketom je, že niektoré staršie prehliadače ho nepodporujú. Ďalší problém sú firewally, z ktorých väčšina môže blokovať komunikáciu a neumožní WebSocketu vytvoriť spojenie. Ak by sme sa teda rozhodli použiť čisto WebSocket protokol, musíme rátať s tým, že naša aplikácia nemusí fungovať správne na všetkých zariadeniach. Modul Socket.io rieši tento problém. Zariadenia, ktoré podporujú WebSocket budú fungovať naďalej bez zmeny a pre tie, ktoré ho nepodporujú, sa Socket.io bude snažiť nájsť najlepšiu možnú alternatívu v nasledujúcom poradí:

1. WebSocket
2. FlashSocket
3. XHR long polling
4. XHR multipart streaming
5. XHR polling
6. JSONP polling
7. iframe

Máme teda zaručené, že aplikácia bude fungovať aj v starších prehliadačoch. Socket.io poskytuje aj API pre Node.js, ktoré vyzerá a používa sa takmer identicky ako na strane klienta. [12, 14]

2.4.4 Passport.js

V našej aplikácii potrebujeme autentizovať užívateľov. Prístup do webového rozhrania aplikácie môže mať len vyučujúci. Passport.js je vhodným modulom do Node.js aplikácií dobre integrovateľný s Express.js frameworkom. Podporuje viacero metód autentizácie, napríklad aj OAuth2, Auth0... My pre jednoduchosť využijeme štandardný prístup pomocou mena a hesla. Kvalitná dokumentácia [2] tohto modulu nám pomohla pri implementácii.

Kapitola 3

Implementácia

V tejto kapitole uvedieme dôležité a zaujímavé časti z kódu aplikácie. Bližšie sa pozrieme na prihlasovanie do aplikácie, spracovávanie dát, vizualizovanie priebehu cvičenia, spôsoby akým je možné hodnotiť riešenia a detegovanie rôznych spôsobov riešenia pomocou K-means.

3.1 Autentizácia

Prihlasovanie do systému je riešené klasicky pomocou mena a hesla. Potrebovali sme zabezpečiť, aby si účet vedeli vytvoriť iba vyučujúci. Spravili sme teda script, ktorý vie spustiť administrátor aplikácie z terminálu. Ten si postupne pýta základné údaje, ktoré sa po vyplnení uložia do databázy. Po prihlásení do webového rozhrania vie užívateľ vytvárať ďalšie účty.

Pred ukladaním nového užívateľa do databázy je potrebné aby heslo bolo bezpečne zahašované. Databázové modely definované pomocou knižnice *Sequelize* podporujú takzvané hooky. Tie sú volané napríklad pred vytvorením záznamu v databáze, po vytvorení, pred/po aktualizovaní alebo mazaní dát... My využívame hook pred vytvorením, ktorý nahradí heslo do zahašovanej podoby. Na hašovanie používame funkciu *bcrypt* z knižnice *bcrypt-nodejs*.

```
User.beforeCreate(function(user, options) {
    return crypt_password(user.password).then(function(hash) {
        user.password = hash;
    }).catch(function(err) {
        if (err) console.error(err);
    });
});
```

Po každej akcii sa aktualizuje exspirácia takzvaného *session_id* na 24 hodín a uloží sa v databáze. Ten zabezpečuje, aby sa užívateľ mohol prihlásiť aj po reštartovaní

servera, čo sme využili pri vývoji systému.

3.2 Spracovávanie dát z GTA aplikácie

Dáta z GTA aplikácie prichádzajú POST požiadavkami na preddefinovaný koncový bod `/gta`. Keďže sme nechceli zahlcovať kompilovaný GTA program odosielaním týchto požiadaviek na server, potrebovali sme zvoliť alternatívu. Do úvahy prichádzali Linuxové príkazy *wget* a *curl*. Na základe porovnania [13] sme sa rozhodli použiť *wget* [7], keďže na školských počítačoch nie je *curl* nainštalovaný a na fungovanie potrebuje knižnicu *libcurl*, čo by znamenalo ďalšiu závislosť aplikácie.

GTA aplikácia si pri štarte spúšťa nový interaktívny bash shell, ktorý si prečíta konfiguračný súbor `.bashrc`. V ňom je nadefinovaná funkcia `prompt_command` a nastavená do premennej prostredia `PROMPT_COMMAND`. Tá je volaná vždy predtým, ako bash vypíše takzvaný prompt - v našom prípade využívame túto funkciu, keď študent napíše nejaký príkaz do terminálu a potvrdí ho enterom. Vtedy odošleme tento príkaz na server *wget*-om. Dáta posielame vo formáte JSON. Znaky v položke *command* sme kódovali pomocou funkcie `urlencode`, ktorá zabezpečovala, že znaky ako medzera nám nepokazili fungovanie odosielania. Niektoré UTF-8 znaky boli zle kódované, čo spôsobovalo chyby na strane servera pri spracovávaní dát funkciou `JSON.parse`. Preto sme sa rozhodli použiť kódovanie pomocou `base64`.

Aby server vedel jasne rozpoznať typ dát, je potrebné nastaviť hlavičku na `Content-Type: application/json`. Prepínač `--no-check-certificate` používame na testovacie účely. Zabezpečuje, že aj keď na server nasadíme vlastnoručne podpísaný (self-signed) TLS certifikát, komunikácia so serverom nebude zamietnutá. V ostrej prevádzke odporúčame tento prepínač vypnúť a používať plne platné certifikáty. Súbor `sample_requests.sh` obsahuje ukážky a simulácie všetkých typov dát, ktoré odosiela GTA aplikácia.

```
wget --no-check-certificate \
     --header="Content-Type: application/json" \
     --post-data '{"type": "start", "user": "uzivatel", \
        "hostname": "pocitac'", "date": "datum", \
        "exercise_number": "id cvicenia", "ip": "ip"}' \
     $SERVER/gta -O /dev/null
```

Pred samotným spracovaním sa kontroluje, či je IP adresa obsiahnutá v položke **ip** povolená (sekcia 3.2.1). Následne sa kontroluje, či je v čase odoslania požiadavky niektorá inštancia z cvičení aktívna a zároveň má rovnaké *id* ako to v požiadavke. Ak sú všetky tieto podmienky splnené, dáta sa uložia do tabuľky *Post*. V opačnom prípade sa zahodia.

Okamžite po uložení potrebujeme odoslať tieto dáta všetkým prihláseným vyučujúcim. To nám zabezpečí funkcia `io.emit`. Výhodou používania Socket.io je, že si môžeme nadefinovať vlastné udalosti. Avšak ich meno nesmie kolidovať s už preddefinovanými udalosťami. Dáta posielame prehľadne uložené v JSON objekte vlastnou udalosťou `new_post`.

```
socketapi.io.emit('new_post', data);
```

Na strane klienta počúvame na túto udalosť a následne zavoláme obsluhovaciu funkciu, ktorá zabezpečí správne zobrazenie v prehliadači.

```
socket.on('new_post', function(data) {
    exercise.new_post(data);
});
```

Každá inštancia cvičenia prechádza tromi rôznymi stavmi.

1. *scheduled* (naplánovaná)
2. *active* (prebiehajúca)
3. *done* (ukončená)

Pri vytváraní týchto inštancií v administrátorskom rozhraní je nutné zadať čas začiatku a konca cvičenia. Na preklápanie inštancie z jedného stavu do nasledujúceho využívame knižnicu *node-schedule*. Tá ponúka definovanie takzvaných *cronjob*-ov, ktoré sú vykonávané periodicky. My sme toto volanie nastavili na každú minútu. Zavolá sa funkcia, ktorá preverí, či je možné preklopiť stav niektorej z inštancií na nasledujúci.

```
schedule.scheduleJob("* * * * *", check_state);
```

3.2.1 Filtrovanie požiadaviek

Občas chce vyučujúci zabezpečiť, aby boli študenti nútení riešiť úlohy iba z konkrétneho miesta (napríklad učebňa v škole). Každá požiadavka z GTA aplikácie nesie so sebou informáciu o IP adrese počítača. Na základe toho vieme určiť, z akej siete táto požiadavka prišla. Vyučujúci si môže nastaviť povolené siete, z ktorých bude možné riešiť cvičenie. Tieto sa dajú nastaviť v súbore `config/app_config.js` v položke `allowed_subnets`. Jednotlivé siete treba zapísať vo formáte `"sieť/maska"` (príklad `"158.195.252.0/23"`) a oddeliť čiarkou. V prípade, že neuvedieme žiadnu povolenú sieť, systém automaticky prepúšťa a spracováva všetky dáta.

Na prácu s IP adresami sme použili knižnicu *ip*. Pri inicializácii aplikácie sa pre každú povolenú sieť vytvorí inštancia `ip.cidrSubnet("ip/maska")`. Vďaka nej môžeme využiť metódu `contains("ip")`, ktorá zistí, či daná IP adresa patrí do siete. Nasledujúca ukážka demonštruje logiku kontroly IP adries.

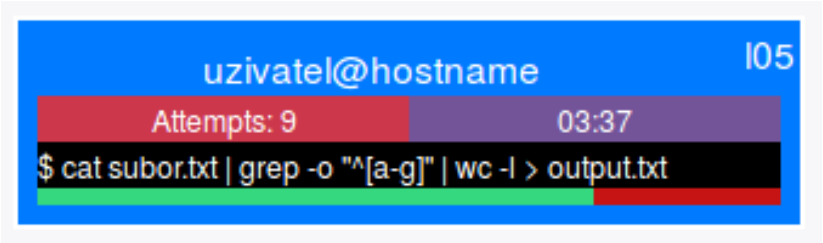
```
function is_ip_allowed(ip)
{
    if (!global.ALLOWED_SUBNETS.length)
        return true;

    for (var i = 0; i < global.ALLOWED_SUBNETS.length; i++) {
        if (global.ALLOWED_SUBNETS[i].contains(ip))
            return true;
    }
    return false;
}
```

3.3 Vizualizácia

Vyučujúcemu potrebujeme zobrazovať o každom študentovi iba najdôležitejšie informácie. Preto sme si navrhli prehľadnú „krabičku“ (obrázok 3.1). V prvom riadku zobrazujeme používateľské meno študenta a pre lepšiu orientáciu v učebni aj takzvaný *hostname*, ktorý určuje meno počítača, na ktorom študent pracuje. V pravom hornom rohu je zobrazený aktuálny názov úlohy, ktorú študent rieši. Druhý riadok je rozdelený na dve položky. Vľavo zobrazujeme koľkými pokusmi sa študent doposiaľ snažil vyriešiť aktuálnu úlohu. Druhá položka počíta čas od prijatia posledného riešenia.

Neaktívni študenti sú takí, ktorí dlhšiu dobu neodošlú pokus o riešenie úlohy. Môže to byť zapríčinené tým, že nerozumejú zadaniu úlohy, nevedia ju riešiť alebo len príliš dlho rozmýšľajú nad riešením. Taktiež príliš veľa pokusov o riešenie úlohy indikuje zaseknutie alebo len drobnú chybu v riešení. V prípade, že počet pokusov alebo čas od posledného riešenia presahuje nakonfigurovaný limit, začne blikať červený indikátor, ktorý upozorní vyučujúceho na zaseknutie a neaktivitu študenta v aktuálnej úlohe. Prvotne sme počítali čas neaktivity na serveri. Uvedomili sme si, že na strane klienta máme taktiež informáciu o čase od posledného riešenia. Preto sme sa rozhodli nezaťažovať server a ponechať detekciu neaktivity na klientskej strane. Tam sa periodicky každých päť sekúnd spúšťa funkcia, ktorá každému študentovi aktualizuje čas neaktivity.

Nižšie, v riadku s čiernym pozadím, sa nachádza posledný príkaz, ktorým sa študent snaží vyriešiť úlohu. V prípade neúspešného riešenia môže vyučujúci nahliadnuť na



Obr. 3.1: Vizualizácia pokroku študenta na cvičení.

tento príkaz, zistiť, kde robí študent chybu, ísť k nemu osobne a poradiť. Úplne naspodu je ukazovateľ pokroku (progress bar), ktorý zobrazuje koľko úloh chýba študentovi vyriešiť do úspešného konca. Jeho dĺžka sa počíta z vopred nastavenej koncovej úlohy.

Takáto jednoduchá vizualizácia nám dáva prehľad s najdôležitejšími informáciami, ktoré pomáhajú vyučujúcemu v lepšej a rýchlejšej orientácii medzi študentmi na cvičení. O každom študentovi máme však zaznamenanú aktivitu počas celého cvičenia. Kliknutím na „krabičku“ sa zobrazí detailný pohľad priebehu cvičenia konkrétneho študenta.

3.4 Hodnotenie riešení

Po skončení cvičenia sa zobrazí tabuľka všetkých zúčastnených študentov s nasledujúcimi údajmi:

- Používateľské meno študenta
- Meno počítača, na ktorom riešil úlohu
- IP adresa počítača
- Posledná úspešne vyriešená úloha
- Čas strávený riešením
- Počet odoslaných príkazov
- Počet bodov za cvičenie
- Počet bonusových bodov za cvičenie

V tomto zozname vieme vyhľadávať a taktiež ho aj zoradiť podľa jednotlivých položiek. Po kliknutí na jednotlivé riadky sa zobrazí okno s riešeniami študenta, v ktorom môže vyučujúci pridať hodnotenie úloh a komentár k nemu. Chceli sme tento proces zautomatizovať a ušetriť vyučujúceho od zbytočného manuálneho hodnotenia. Pridali sme teda tlačítko, ktoré pošle na server požiadavku automaticky obodovať riešenia.

Pri vytváraní inštancie cvičenia je k dispozícii možnosť nastavenia bodov za konkrétne úlohy. Vzhľadom k tomu, že úloh môže byť veľmi veľa, by bolo nepraktické nastavovať zlomok bodov ku každej jednej. Rozhodli sme sa preto, že meno úlohy bude možné zadať aj ako regulárny výraz.

Príklad. Majme úlohy označené ako *l01*, *l02*, *l03*, ... , *l0n*. Nech za každú jednu vyriešenú úlohu je jeden bod. Potom stačí nastaviť regulárny výraz `l[0-9]+` a počet bodov na 1.

Algoritmus automatického hodnotenia si najskôr vyberie z databázy všetky úspešné riešenia. Postupne sa tieto riešenia prechádzajú od najnovšieho po najstaršie. Každé sa porovná so všetkými regulárnymi výrazmi, ktoré boli prednastavené. Ak položka *level* zodpovedá niektorému z týchto regulárnych výrazov, potrebujeme ešte overiť MD5 haš, ktorý vygenerovala GTA aplikácia a odoslala spolu s riešením.

Vstupmi do tohto hašu sú 3 hlavné položky: *názov cvičenia*, *názov úlohy* a *domovský priečinok študenta*. Tie sú ešte skombinované s písmenkami **i**, **j**, **k** a **l**. V GTA aplikácii vznikla pri programovaní zaujímavá chyba. Pri vytváraní vstupu do MD5 hašovacej funkcie sa očakáva názov úlohy ako celé číslo (integer). Názov úlohy je ale vždy reprezentovaný ako reťazec znakov. Po vložení tohto reťazca do číselného formátu jazyk Go vygeneruje výstup `%!d(string=level)`, kde level je názov úlohy. Výsledným vstupom do MD5 hašovacej funkcie je teda nasledujúci reťazec, kde operátor + znamená zreťazenie.

$$\texttt{"i"} + \textit{názov cvičenia} + \texttt{"j\%!d(string="} + \textit{názov úlohy} + \texttt{")k"} + \textit{domovský priečinok} + \texttt{"l"}$$

Tento nami vygenerovaný haš porovnáme s tým, ktorý vygenerovala aplikácia GTA. Ak sa rovnajú, študentovi pripočítame množstvo bodov, ktoré prislúcha danej úlohe. Podľa toho, či je táto úloha nastavená ako bonus, tieto body pridelíme do stĺpca *bonus* alebo *score*.

Takto ohodnotené riešenia je následne možné exportovať do csv súboru, ktorý je potom ľahko importovateľný do systému Moodle, kde študenti uvidia svoje hodnotenie.

3.5 Detekcia rôznych spôsobov riešenia úloh

Fundamentálny problém, ktorý tvorí významnú rolu v rôznych disciplínach ako sú rozpoznávanie vzorov, spracovanie obrazu, strojové učenie a štatistika, sa nazýva problém zhlukovania. V základe je zhlukovanie definované ako problém hľadania homogénnych skupín z danej množiny dát pomocou metódy bez učiteľa (unsupervised learning). Každá z týchto skupiniek nesie názov *cluster*, ktorý definujeme ako región s lokálne hustejším výskytom objektov ako v iných regiónoch.

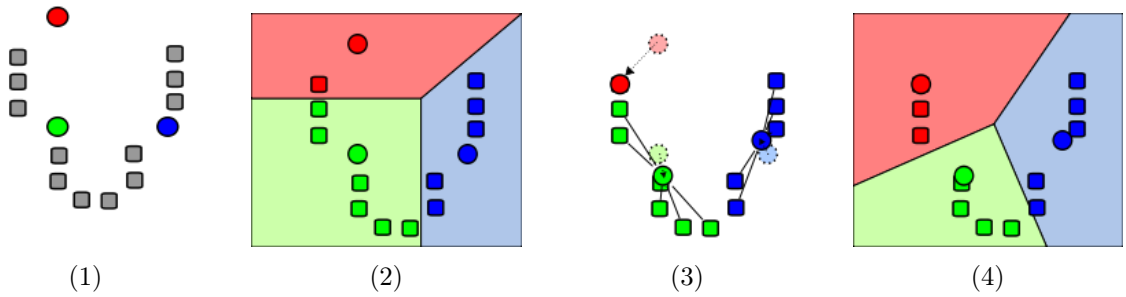
V tejto práci sme sa rozhodli poskytnúť vyučujúcemu prehľad o rôznych spôsoboch riešeniach úloh. Získa tak cennú spätnú väzbu o rozmýšľaní študentov a môže prehodnotiť úpravu prednášky alebo zadaní úloh v budúcnosti. Na túto klasifikáciu riešení sme sa rozhodli použiť metódu strojového učenia *K-means*.

3.5.1 K-means

Najjednoduchšou formou zhlukovania je rozdeľovanie jednotlivých objektov do disjunktných podmnožín tak, že splnia špecifické kritériá. Každá z týchto množín je reprezentovaná takzvaným *centroid*-om, ktorý je stredom (ťažiskom) celého zhluku. Najrozšírenejšie kritérium je *kritérium chyby clustera*, ktoré pre každý jeden objekt spočíta jeho štvorcovú vzdialenosť od centroidu a potom vezme súčet týchto vzdialeností pre celý cluster. Veľmi populárnou metódou, ktorá minimalizuje túto chybu je algoritmus *K-means*. Predpokladá, že objekty, s ktorými pracuje, sú vektory. Je všeobecne známe, že táto metóda dosť závisí od náhodného výberu centroidov na začiatku a niekedy môže vyžadovať viacero reštartov. [9]

Algoritmus funguje v štyroch základných krokoch (vizualizácia na obrázku 3.2):

1. Voľba k počiatočných centroidov
2. Každý vektor z množiny dát je priradený do clustera s najbližším centroidom
3. Každý cluster si prepočíta nový centroid
4. Kroky 2 a 3 sa opakujú pokiaľ aspoň jeden vektor zmení cluster alebo sa prekročí maximálny povolený počet iterácií



Obr. 3.2: Vizualizácia fungovania K-means (zdroj: wikipedia, Weston.pace, licencia CC BY-SA 3.0)

Najviac používané metódy pri voľbe počiatočných k centroidov sú *Forgyho* a *Random Partition*. Prvá spomínaná si z celej množiny dát vyberie k náhodných vektorov, kde každý bude počiatočným centroidom a pokračuje krokom tri. Druhá priradí na

začiatku každý objekt do niektorého z clusterov.

Formálnejšie, majme množinu dát $S = \{x_1, x_2, \ldots, x_n\}$. Nech k je počet clusterov, do ktorých chceme dáta prerozdeliť. Ďalej nech $m_1^{(1)}, m_2^{(1)}, \ldots, m_k^{(1)}$ je k náhodných centroidov na začiatku algoritmu.

Krok priradenia:

$$S_i^{(t)} = \left\{ x_p \mid \forall x_p \in S \; \forall j, 1 \leq j \leq k : d\big(x_p, m_i^{(t)}\big) \leq d\big(x_p, m_j^{(t)}\big) \right\}$$

kde d je funkcia vzdialenosti medzi dvoma bodmi a každé x_p je v iterácii t priradené do práve jedného $S_i^{(t)}$

Krok prepočtu centroidov:

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

Na konci výpočtu dostaneme rozdelenie dát do clusterov. Je dobré vyskúšať rôzne hodnoty k kvôli presnejšej klasifikácii. V tejto práci sme použili na klasifikáciu jednotlivých riešení knižnicu *node-kmeans*, ktorá je stabilná a má dobrú podporu. Na inicializáciu centroidov využíva *Forgyho* metódu.

3.5.2 Reprezentácia vstupu

V databáze máme uložené príkazy jednotlivých riešení v textovej podobe. Ďalej pracujeme iba s úspešnými riešeniami. Klasifikátor očakáva ako vstupné dáta vektory. Každý príkaz rozparsujeme na slová. Pomáha nám v tom knižnica *node-shell-quote*. V Linuxe existujú takzvané premenné prostredia, ktoré sa pri interpretovaní volajú s prefixom **$**. Naša knižnica taktiež podporuje interpretovanie týchto premenných, avšak my to využívať nepotrebujeme. Počas používania tejto knižnice sme narazili na problém, kde regulárny výraz obsahoval identifikátor konca riadku **$**. Funkcia na parsovanie sa ho snažila interpretovať ako premennú. Keďže nenašla žiadnu nastavenú hodnotu, tak túto premennú interpretovala ako prázdny reťazec, čo spôsobilo premazanie znaku **$**. Preto sme sa rozhodli opraviť túto chybu vo vlastnom klone tejto knižnice a ponechávať tento znak v reťazci pokiaľ sa nenájde prislúchajúca hodnota.

Príklad. Majme príkaz
`cat file.txt | grep "^ab[0-9]+" | grep -n -o "[0-9]+" > output.txt`.
Ten nám knižnica rozparsuje na nasledujúci zoznam:

```
["cat", "file.txt", {op: "|"}, "grep", "^ab[0-9]+", {op: "|"},
"grep", "-n", "-o", "[0-9]+", {op: ">"}, "output.txt"]
```

My si tento výstup prerobíme na množiny slov. Rozhodli sme sa pre unigramy a špeciálne bigramy, ktoré definujeme nižšie. Používateľ si môže vybrať, ktorú reprezentáciu vstupu chce.

Unigramy: Sú to slová, ktoré dostaneme z parsera. Z nich spravíme množinu slov, v ktorej sa každé slovo bude nachádzať maximálne raz.

$$\{\texttt{cat},\ \texttt{file.txt},\ \texttt{|},\ \texttt{grep},\ \texttt{\^{}ab[0-9]+},\ \texttt{-n},\ \texttt{-o},\ \texttt{[0-9]+},\ \texttt{>},\ \texttt{output.txt}\}$$

Špeciálne bigramy: Bigramy sú dvojice položiek, ktoré sa v texte nachádzajú bezprostredne za sebou. Môžu to byť fonémy, slabiky, písmená alebo slová. V našom prípade sa môže stať, že rovnaký prepínač je použitý v dvoch rôznych podpríkazoch, kde zastáva rozdielne funkcionality. Preto sme sa rozhodli spraviť bigramy, kde prepínače a argumenty spojíme do dvojice s príkazom, a operátory a samotné mená príkazov ostanú ako unigramy. Z hore uvedeného príkazu v príklade teda dostaneme nasledujúcu reprezentáciu:

$$\{\texttt{cat},\ \texttt{cat file.txt},\ \texttt{|},\ \texttt{grep},\ \texttt{grep \^{}ab[0-9]+},$$
$$\texttt{grep -n},\ \texttt{grep -o},\ \texttt{grep [0-9]+},\ \texttt{>},\ \texttt{output.txt}\}$$

Túto množinovú reprezentáciu spravíme pre každé jedno riešenie. Následne potrebujeme spraviť zjednotenie slov cez všetky tieto rozparsované slová. Formálne, nech $R_1, R_2, \ldots, R_n$ sú množiny slov. Potom množina všetkých slov, ktoré sa vyskytli v riešeniach je množina $R = \bigcup_{i=1}^{n} R_i$. Predpokladajme ďalej, že jej prvky majú svoju pevnú pozíciu a vieme ich indexovať.

Teraz potrebujeme každú jednu množinu s riešením reprezentovať ako vektor s hodnotami 0 a 1. Hodnota 1 označuje, že slovo z množiny R sa nachádza v riešení a 0 ak sa nenachádza. Nech výsledné vektory riešení sú $\vec{\alpha_1}, \ldots, \vec{\alpha_n}$. Potom ich dimenzia je $d(\vec{\alpha_i}) = |R|$ pre každé $i = 1, \ldots, n$. Každý vektor má na súradniciach hodnoty $\vec{\alpha_i} = (a_1, \ldots, a_{|R|})$, pre ktoré platí:

$$a_j = \begin{cases} 1, & \text{ak sa slovo z } R \text{ na } j\text{-tej pozícii nachádza v } R_i, \\ 0, & \text{inak} \end{cases}$$

Dostali sme teda vektorovú reprezentáciu jednotlivých riešení, ktoré môžeme použiť ako vstup do algoritmu *K-means*.

3.5.3 Meranie vzdialenosti

K-means potrebuje počítať vzdialenosť medzi jednotlivými vektormi, aby ich vedel následne rozdistribuovať do clusterov podľa najbližších centroidov. Väčšinou sa používa *euklidovská vzdialenosť*. Spočiatku sme chceli používať *Levensteheinovu* alebo *Hammingovu* vzdialenosť. Tieto však pracujú so vstupom ako celkom. Prepínače v Linuxe

môžu byť rôzne preusporiadané, čo by pri týchto metódach vypočítalo veľké vzdialenosti. Preto sme sa rozhodli použiť *Jaccardovu* a *kosínusovú* vzdialenosť.

Jaccardova vzdialenosť: Jaccardov index, taktiež známy ako Jaccardov koeficient podobnosti, je koeficient, ktorý určuje podobnosť medzi dvoma konečnými množinami a je definovaný ako veľkosť prieniku vydelený veľkosťou zjednotenia.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Ak sú množiny A aj B prázdne, potom definujeme $J(A,B) = 1$. Jaccardova vzdialenosť, ktorá meria rozdielnosť medzi množinami, je komplementárna k Jaccardovmu indexu.

$$d_J(A,B) = 1 - J(A,B)$$

Je nutné spomenúť, že Jaccardova vzdialenosť pracuje s množinovými operáciami. Knižnica *node-kmeans* počíta centroidy priemerovaním súradníc (viď sekciu 3.5.1). Na základe toho sme sa rozhodli spraviť hranice, kde ak je hodnota danej súradnice väčšia alebo rovná 0.5, slovo patrí do centroidu, v opačnom prípade nepatrí.

Kosínusová vzdialenosť: Kosínusová podobnosť je meranie podobnosti medzi dvoma nenulovými vektormi z unitárneho priestoru, ktorá počíta kosínus uhla, ktorý zvierajú. Tým pádom nezáleží na dĺžke týchto vektorov.

$$C(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} = \frac{\sum_{i=1}^{|R|} a_i b_i}{\sqrt{\sum_{i=1}^{|R|} a_i^2} \sqrt{\sum_{i=1}^{|R|} b_i^2}}$$

Táto podobnosť sa spravidla počíta v priestore s nezápornými hodnotami na súradniciach, kde výsledok je ohraničený v intervale $[0, 1]$. Všimnime si, že dva vektory sú si maximálne podobné, ak sú paralelné a naopak maximálne odlišné, ak sú na seba kolmé. Kosínusová vzdialenosť je komplementárna ku kosínusovej podobnosti.

$$d_C(A,B) = 1 - C(A,B)$$

Tieto metódy sme použili ako základné metriky na počítanie vzdialenosti medzi dvoma riešeniami.

3.5.4 Vizualizácia n-rozmerných vektorov v 2D priestore

Pre lepšiu predstavivosť o vzdialenosti riešení, ktoré vstupujú do *K-means* algoritmu potrebujeme vizualizovať jednotlivé vektory v 2D priestore. Jediným problémom je, že tieto vektory majú veľké dimenzie, ktoré nemôžeme priamo priradiť na $[x, y]$ pozíciu.

Rozhodli sme sa využiť algoritmus strojového učenia *t-SNE* (t-distributed stochastic neighbor embedding) určený na vizualizáciu, ktorý vyvinuli Laurens van der Maaten a Geoffrey Hinton. Je to technika na nelineárnu redukciu dimenzie veľkorozmerných dát $X = \{x_1, x_2, \ldots, x_n\}$ na body v 2D alebo 3D priestore $Y = \{y_1, y_2, \ldots, y_n\}$, ktoré vieme vizualizovať. Základom redukcie dimenzií je modelovať objekty taký spôsobom, že objekty, ktoré sú vo veľkorozmernom priestore veľmi vzdialené ostanú vzdialené a tie, ktoré sú blízko seba ostanú blízko. Tým máme zachovanú podobnosť. [10]

Algoritmus *t-SNE* spúšťame vo webovom prehliadači pomocou knižnice *tsnejs*.

Kapitola 4

Testovanie

V tejto kapitole zhrnieme výsledky z testovania aplikácie na cvičeniach kurzu Linux pre používateľov a ukážeme si možnosti vizualizovania zozbieraných riešení našou aplikáciou.

4.1 Priebeh testovania a funkčnosť systému

Systém sme nainštalovali na Linuxový server. Pred každým cvičením vyučujúci vytvoril inštanciu cvičenia, nastavil interval jeho trvania, body za úspešné riešenia. Spolu sa aplikácia testovala na desiatich cvičeniach. Študenti si stiahli z Moodle GTA skript, ktorý si následne spustili na počítačoch v učebni.

Počas cvičenia nenastávali žiadne zdržania aplikácie, ktoré by spôsobili oneskorenú vizualizáciu na strane klienta. Na poslednom cvičení sa vyskytla chyba pri spracovávaní tela POST požiadavky, kedy GTA aplikácia zakódovala nesprávne riešenie a na strane servera ho funkcia `JSON.parse` nevedela spracovať. Preto sme sa nakoniec rozhodli nahradiť `urlencode` metódou `base64`.

4.2 Zhodnotenie aplikácie vyučujúcimi

Na konci semestra sme požiadali vyučujúcich o zhodnotenie aplikácie a jej prínos na cvičeniach.

Postrehli, že v porovnaní s minulým rokom sa efektivita zásadne zvýšila, nakoľko je možné preemptívne zasiahnuť a zastaviť sa pri študentovi, o ktorom je jasné, že je zaseknutý, namiesto toho, aby museli čakať, kedy sa prihlási. Tiež sa dá lepšie odhadnúť, kedy približne cvičenie skončí, pretože aplikácia zobrazuje, v ktorom leveli sa nachádzajú jednotliví študenti.

Druhým najvýraznejším prínosom aplikácie je možnosť klasifikovať riešenia. Vyučujúcim to prinieslo lepší pohľad na rozmýšľanie študentov. Prednášajúcemu zas možnosť

sústrediť sa v prednáške na študentmi nepochopené učivo.

Vyučujúci taktiež hodnotia pozitívne implementáciu príkazu `gta_help`, vďaka ktorej mohli študenti upozorniť vyučujúcich a požiadať o pomoc. My máme takto záznamy v databáze a mohli by sme následne vyhodnotiť, pri ktorých úlohách študenti žiadali o pomoc najviac. Avšak, tento príkaz mohli študenti používať až od polovice semestra. Preto boli zvyknutí sa hlásiť klasicky zdvihnutím ruky a tento príkaz využívali veľmi sporadicky. Z toho dôvodu nemáme relevantné dáta k vyhodnoteniu potreby pomoci v jednotlivých úlohách.

4.3 Výsledky

Po skončenom cvičení máme prehľad o všetkých príkazoch, ktoré študenti odoslali. Z nich vieme následne vyrobiť graf percentuálneho zastúpenia úspešných a neúspešných príkazov (obrázky 4.1 a 4.2). Po prejdení myškou cez jednotlivé stĺpce vo webovom rozhraní sa zobrazí aj počet úspešných a neúspešných príkazov. Na štvrtom cvičení použili študenti pri riešení úlohy *l01* 559 príkazov. Na deviatom cvičení to už bolo 1829. Je teda vidieť značné zvýšenie náročnosti. Tabuľka 4.1 zachytáva základný prehľad o trvaní cvičení.

	Počet úloh	Počet študentov	Priemerný počet príkazov	Medián počtu príkazov	Priemerný čas riešenia	Medián času riešenia
Cvičenie 1	7	54	11.52	11	00:19:52	00:21:09
Cvičenie 2	18	54	68.44	61	00:29:49	00:30:42
Cvičenie 3	10	54	76.70	71.50	01:02:17	01:06:55
Cvičenie 4	7	54	69.41	65	00:53:36	00:54:50
Cvičenie 5	6	52	36.53	33	00:53:40	00:53:08
Cvičenie 6	7	54	43.41	43	00:31:08	00:31:09
Cvičenie 7	5	53	52.15	48	00:38:43	00:39:15
Cvičenie 8	6	50	96.60	85	00:54:07	00:55:26
Cvičenie 9	5	51	99.39	97	01:12:21	01:09:47
Cvičenie 10	4	47	54.63	50	01:21:59	01:26:11

Tabuľka 4.1: Vyhodnotenie trvania cvičení

Klasifikácia riešení metódou *K-means* nám priniesla veľmi zaujímavé výsledky. Pri testovaní zhlukovania príkazov sa špeciálne bigramy javia ako lepšia reprezentácia vstupu. Pri príkazoch s nízkym počtom podpríkazov a argumentov stačia aj unigramy.

Niekedy je nutné spustiť algoritmus viackrát kvôli náhodnosti výberu počiatočných centroidov.

Nižšie uvádzame príklady troch správnych riešení úlohy, v ktorej mali študenti v súbore *test_ cases.py* nájsť definíciu funkcie, v ktorej sa nachádza slovo *page*. Každé z riešení sa dostalo do jedného z clusterov. V tabuľke 4.2 môžme vidieť vzdialenosti medzi jednotlivými riešeniami.

(a) `grep -n -i "def.*page[a-z^(]" test_cases.py > funkcia.txt`

(b) `grep -n "def" test_cases.py | grep "page.*(" > funkcia.txt`

(c) `cat test_cases.py | grep -n "def.*" | grep "page(" > funkcia.txt`

	Jaccardova vzdialenosť	Kosínusová vzdialenosť
Unigramy (a), (b)	0.5	0.33184689521893906
Unigramy (a), (c)	0.5454545454545454	0.37005921165128797
Unigramy (b), (c)	0.4545454545454546	0.29289321881345254
Špec. bigramy (a), (b)	0.5	0.33184689521893906
Špec. bigramy (a), (c)	0.6666666666666667	0.49604736932103044
Špec. bigramy (b), (c)	0.5833333333333333	0.41074434901121050

Tabuľka 4.2: Porovnanie vzdialeností reprezentácií vstupov (a), (b), (c)

Poďme sa ďalej pozrieť na úlohu, kde bolo treba vypísať posledný stĺpec zo súboru `/etc/passwd`. Prvé dve riešenia sa dostali do rovnakého clustera.

(1) `grep "[^:]*$" -o /etc/passwd > zoznam.txt`

(2) `grep "[^:]*$" -o /etc/passwd > ../zoznam.txt`

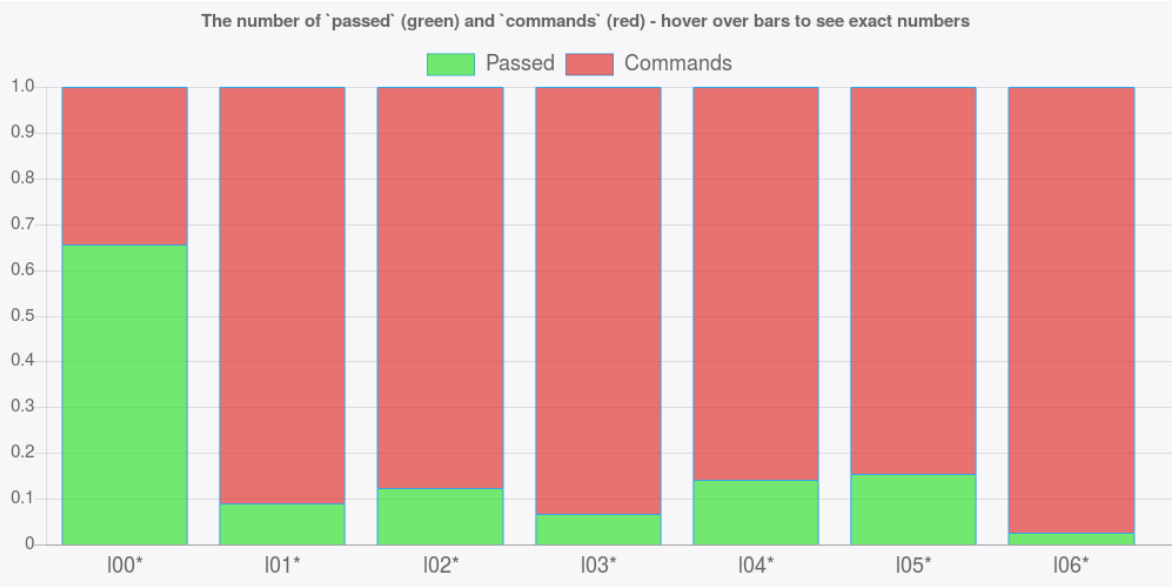
(3) `cat /etc/passwd | grep -o "[^:]*$" > zoznam.txt`

	Jaccardova vzdialenosť	Kosínusová vzdialenosť
Unigramy (1), (2)	0.2857142857142857	0.16666666666666652
Unigramy (1), (3)	0.25	0.1339745962155613
Unigramy (2), (3)	0.4444444444444444	0.2783121635129677
Špec. bigramy (1), (2)	0.2857142857142857	0.16666666666666652
Špec. bigramy (1), (3)	0.5	0.3195861825602282
Špec. bigramy (2), (3)	0.6363636363636364	0.45566894604818264

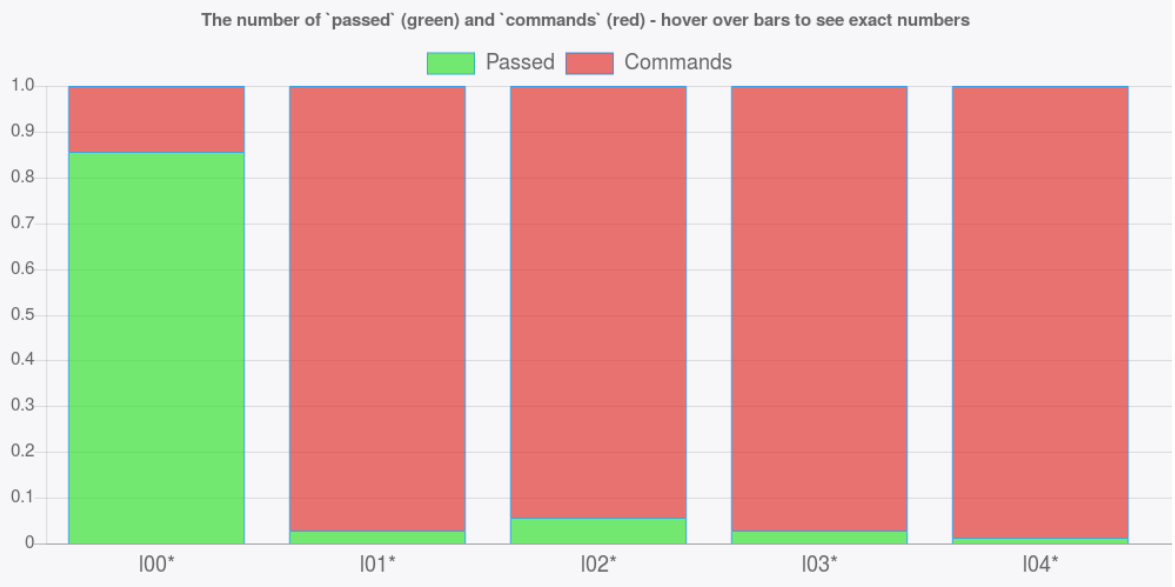
Tabuľka 4.3: Porovnanie vzdialeností reprezentácií vstupov (1), (2), (3)

V tomto prípade si môžeme v tabuľke 4.3 všimnúť, že vzdialenosti unigramových reprezentácií (1) a (2) sú väčšie ako pri (1) a (3). My by sme však chceli, aby to bolo presne opačne a príkazy (1) a (2) boli k sebe bližšie ako (1) a (3), pretože tieto dve riešenia sa od seba odlišujú len v troch znakoch. Nami navrhnuté špeciálne bigramy vyriešili tento problém, čo sa prejavilo aj na vzdialenostiach. Vo všeobecnosti však nemusí platiť, že naše špeciálne bigramy sú vždy lepšie ako unigramy. Taktiež, z výsledkov nie je úplne jasné, ktorá metrika je lepšia. Vidíme iba, že Jaccardova vzdialenosť udáva väčšinou o pár stotín väčšiu vzdialenosť ako kosínusová. Preto sme sa v aplikácii rozhodli ponechať obidve možnosti reprezentácií vstupu a metrík a používateľ sa môže rozhodnúť, ktorú použije.

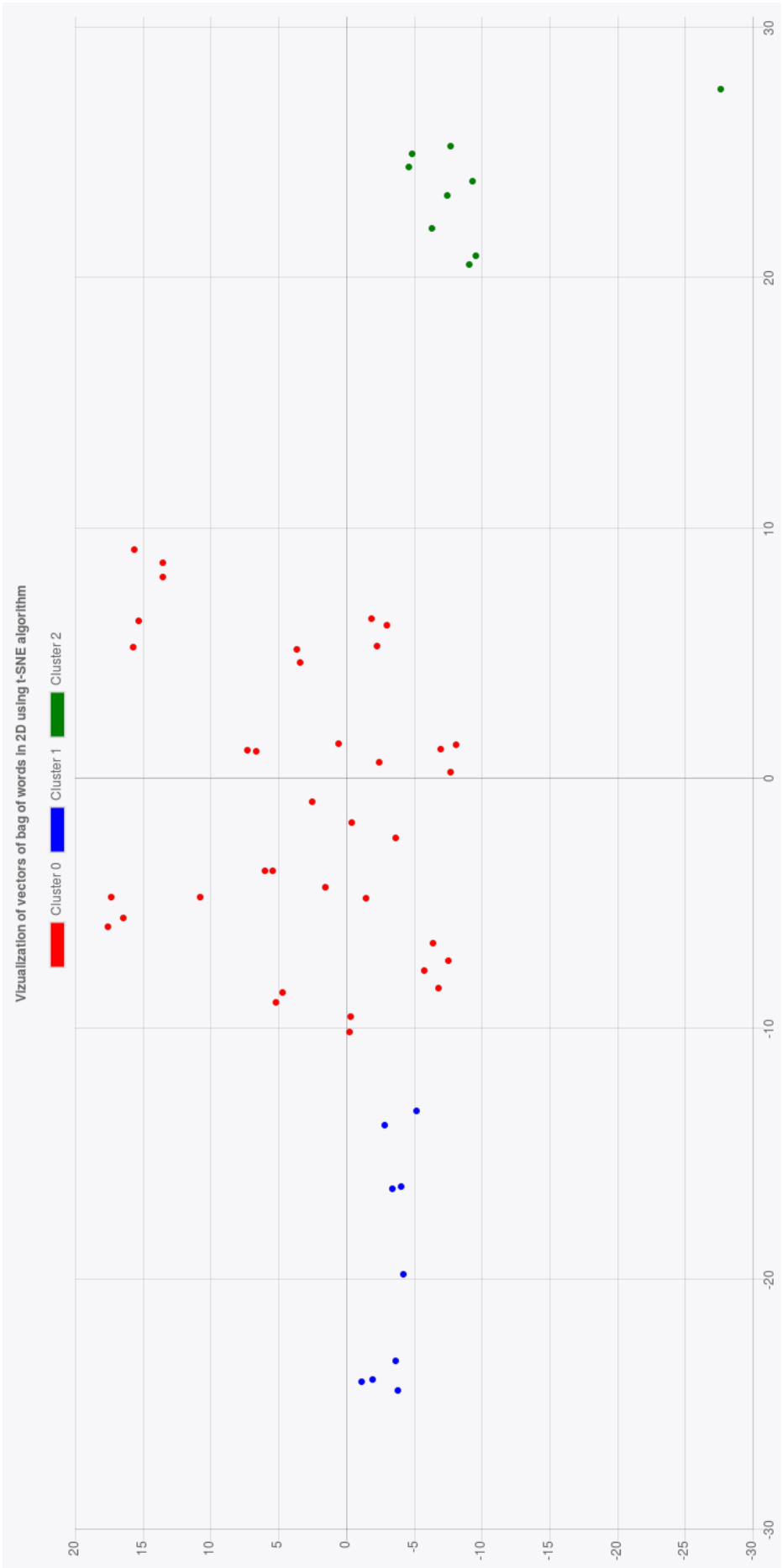
Na obrázku 4.3 je ukážka zobrazenia riešení v 2D priestore pomocou t-SNE, ktoré boli rozdelené do troch clusterov.



Obr. 4.1: Percentuálne zastúpenie úspešných a neúspešných pokusov o riešenie v rámci každej z úloh (cvičenie č. 4)



Obr. 4.2: Percentuálne zastúpenie úspešných a neúspešných pokusov o riešenie v rámci každej z úloh (cvičenie č. 9)



Obr. 4.3: Vizualizácia vzdialeností riešení pomocou t-SNE (3 clustery)

Záver

V tejto práci sa nám podarilo úspešne navrhnúť a implementovať systém, ktorý monitoruje priebeh cvičenia. Systém zbiera dáta od študentov riešiacich úlohy v termináli. Tieto dáta sú následne spracovávané a v reálnom čase zobrazované vyučujúcim. Popri zbieraní týchto dát systém automaticky deteguje kritické situácie ako napríklad zaseknutie študentov na niektorej z úloh alebo príliš veľkú neaktivitu. Vyučujúci sa tak môžu viac venovať týmto študentom. Taktiež sa nám podarilo naprogramovať rozhranie, v ktorom je možné hodnotiť riešenia študentov buď ručne alebo automaticky. Veľkým prínosom je aj možnosť zhlukovania podobných riešení pomocou metódy *K-means*, ktorá nám poskytuje prehľad o rozmýšľaní študentov.

Systém sa podarilo priebežne testovať na desiatich cvičeniach. Počas nich sme získavali spätnú väzbu od vyučujúcich, vďaka čomu sa podarilo implementovať veľa vylepšení. Taktiež sa podarilo odhaliť chyby a pripraviť aplikáciu na plnohodnotné použitie.

Táto aplikácia prináša mnoho nápadov na prácu v budúcnosti. Počas semestra sme na každom cvičení zozbierali značné množstvo dát, ktoré by sa v budúcich rokoch dali využiť. Na základe historických dát vedieť lepšie odhadnúť, koľko by daná úloha mala približne trvať (tak ako v počte pokusov tak aj časovo), nakoľko reálny čas súčasných riešení značne prekračuje vymedzený priestor. Spraviť lepšiu heuristiku na detekciu zaseknutých študentov. Zaradiť študentov do istých „kategórii problémovosti“ vzhľadom na prílišnú podobnosť riešení s ostatnými, či príliš veľkú dobu trvania riešenia cvičenia voči ostatným. Týmto študentom by sa potom mohli vyučujúci lepšie venovať, zvoliť iný prístup, aby učivo rýchlejšie a ľahšie pochopili. Ďalším zaujímavým vylepšením by bolo implementovať distribúciu cvičiacich k jednotlivým študentom. Každý cvičiaci by mal mobilné zariadenie, kam by systém posielal notifikácie. Aplikáciu by sme chceli rozšíriť aj na iné domény výuky (programovanie v Pythone, C++, Scratchi...), čo by vyžadovalo pridanie odosielania dát z testovačov a jemnú modifikáciu nášho systému.

Literatúra

[1] Network monitoring. https://en.wikipedia.org/wiki/Network_monitoring. Získané 2018-03-26.

[2] Passport.js dokumentácia. http://www.passportjs.org/docs. Získané 2017-11-30.

[3] Sequelize dokumentácia. http://docs.sequelizejs.com. Získané 2017-11-29.

[4] Scott W. Ambler. Mapping Objects to Relational Databases: O/R Mapping In Detail. http://www.agiledata.org/essays/mappingObjects.html. Získané 2017-11-29.

[5] Tyler Crawford and Tauqeer Hussain. A Comparison of Server Side Scripting Technologies. In *Int'l Conf. on Software Engineering Research and Practice*, pages 69–76, 2017.

[6] Ian Fette and Alexey Melnikov. The WebSocket Protocol (RFC 6455). https://tools.ietf.org/html/rfc6455, 2011.

[7] Free Software Foundation, Inc. GNU Wget 1.18 Manual. https://www.gnu.org/software/wget/manual/wget.html. Získané: 2017-12-02.

[8] David Herron. *Node.js Web Development (third edition)*. Packt Publishing Ltd., 2016.

[9] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[10] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[11] Nick Montfort and Paulo Urbano. A quarta Era da Ficção Interactiva. *Nada*, 8, 2006.

[12] Rohit Rai. *Socket.IO Real-time Web Application Development*. Packt Publishing Ltd., 2013.

[13] Daniel Stenberg. curl vs Wget. `https://daniel.haxx.se/docs/curl-vs-wget.html`. Získané: 2017-12-02.

[14] David Walsh. WebSocket and Socket.IO. `https://davidwalsh.name/websocket`. Získané 2018-03-30.

[15] Marek Šuppa and Ondrej Jariabka. go-term-adventures: Learning the *nix Command line, text adventure style. In B. Brejová, J. Guričan, and T. Vinař, editors, *Proceedings of the Student Science Conference 2017*, page 245, FMFI UK, Bratislava, 2017. CreateSpace Independent Publishing Platform.

Appendix A

Príloha A obsahuje zdrojový kód aplikácie, ktorý je dostupný aj na portáli GitHub na adrese `https://github.com/NaiveNeuron/gta-monitor`. Súbor `README.md` popisuje proces inštalácie a spustenia aplikácie. Skript `sample_requests.sh` simuluje odosielanie požiadaviek GTA aplikácie. Všetky tieto súbory možno nájsť na priloženom CD.