Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Vývojové prostredie pre online programovanie v skupine

Bakalárska práca

Bratislava, 2015

Jozef Dúc

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Vývojové prostredie pre online programovanie v skupine

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Ing. František Gyárfáš, PhD.

Bratislava, 2015

Jozef Dúc

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

. . . . . .

Ďakujem

Abstrakt

V tejto bakalárskej práci vytvárame návrh vývojového prostredia pre online programovanie v skupine. Prostredie umožňuje zdieľané editovanie zdrojového kódu a jeho testovanie pomocou metodológie Test-driven development. Súčasťou je administrátorske rozhranie zadávateľa úloh.

V práci taktiež analyzujeme použité technológie a podobné existujúce riešenia. Jadro práce tvorí samotný návrh, špecifikujeme požiadavky na systém. Našu implementáciu následne popisujeme v kapitole Implementácia riešenia, kde uvádzame aj krátku inštalačnú príručku.

V závere hodnotíme našu aplikáciu a popisujeme jej testovanie na FMFI UK.

Kľúčové slová: programovanie v skupine, zdieľané editovanie zdrojového kódu, Test-driven development

Abstract

In this bachelor's thesis we create Online environment for group programming. The environment includes shared code editing and its testing through Test-Driven Develoment methodology. Administrator interface for the task-giver is also included.

The part of the thesis is analysis of used technology and existing systems. The base of thesis is the software proposal and the specifications of system requirements. Our implementation is described in the chapter Implementation, where we attached a brief installing tutorial.

At the end of the thesis we summarize our application and its testing at Faculty of mathematics, physics and informatics in Comenius University.

Key words: group programming, share code editing, Test-driven development

Obsah

Zoznam obrázkov

Zoznam skratiek

API	Application Programming Interface
GUI	Graphical User Interface
IDE	Integrated Development Environment
TDD	Test Driven Development
WYSIWYG	What You See Is What You Get

Kapitola 1

Úvod

1.1 Motivácia

Prvýkrát som sa s myšlienkou programovania v skupine stretol v prvom ročníku, kde sa cvičiaca takýmto spôsobom snažila oživiť cvičenia. Ako "vývojové prostredie"sme používali Google dokumenty, čo malo mnoho nevýhod počnúc chýbajúcim syntax highlighting a končiac tým, že náš kód sme si nemohli skompilovať a overiť jeho správnosť. Museli sme ho skopírovať do iného prostredia, čo celý proces spomalovalo a zneefektívňovalo.

Skúsenosť to však bola zaujímavá a toto cvičenie si dodnes pamätám ako jedno z najlepších na vysokej škole. Všetci sme mali pred sebou rovnaký kód, o ktorom sme diskutovali a spoločne sme sa učili, čo to je backtracking.

Druhýkrát som si programovanie v skupine (aj keď v menšej - tvorenej dvomi ľuďmi) vyskúšal na extrémnom programovaní. Pôvodná idea je dvaja ľudia zdieľajúci jednu klávesnicu, no praktickejšie je zdieľanie rovnakého kódu v reálnom čase.

Programovanie sa považuje za individuálnu činnosť, avšak neplatí to vždy. Výnimkou sú už spomínané didaktické účely - vyučovanie základov programovania a extrémne programovanie, no aj v iných situáciách môže byť zdieľané editovanie zdrojového kódu veľkou výhodou. A práve preto sme sa rozhodli vytvoriť vlastné vývojové prostredie pre online programovanie v skupine.

1.2 Cieľ

Cieľom tejto bakalárskej práce je navrhnúť a vytvoriť také vývojové prostredie, ktoré spája funkcionalitu viacerých nástrojov do jednej aplikácie.

Najdôležitejšou časťou aplikácie je zdieľané editovanie zdrojového kódu (inšpirácia: Google Docs) a jeho následné testovanie na serveri (inšpirácia: školská stránka Františka Gyárfáša).

Keďže primárne použitie našej aplikácie je určené pre vysokú školu, naším cieľom je implementovať ju ako úlohový systém (inšpirácia: L.I.S.T) s administrátorským rozhraním pre učiteľa resp. zadávateľa úloh.

Študenti majú možnosť vytvárať skupiny, v ktorých môžu následne písať program, testovať ho a pridávať vlastné testy. Jednou z funkcionalít je aj verzionovanie kódu (inšpirácia: GitHub).

Kapitola 2

Východiská

Táto kapitola sa zaoberá teoretickým pozadím našej práce. Vysvetľujeme v nej pojmy súvisiace s aplikáciou, hodnotíme a analyzujeme existujúce riešenia a na záver popisujeme technológie, ktoré sú použité v našej implementácii.

2.1 E-learning

E-learning je moderný spôsob vyučovania s využitím informačných technológií. Je to vzdelávací proces, ktorý slúži na tvorbu zadaní, kurzov a následného testovania žiakov. E-learning má veľký potenciál vo výučbe programovania, vzhľadom na to, že je úzko spojený s informačnými technológiami a priamo ich využíva. Vo svete vznikajú špecializované e-learningové kurzy, ktoré prevedú užívateľov od základov programovania, cez rôzne druhy programovacích jazykov, algoritmov a dátových štruktúr až k pokročilým metódam v programovaní.

2.2 Test Driven Development

V programovaní je *Test Driven Development*[Bec12] známa agilná metóda. Pomocou tejto metódy sa testujú časti aplikácie ako sú moduly, procedúry, funkcie a iné časti zdrojového kódu, aby sa overilo, že pracujú tak, ako sa od nich očakáva. Celý proces testovania prebieha v nasledujúcich šiestich krokoch:

Pridanie alebo prepísanie testu: pred začatím nového testovania je potrebné pridať nové testy alebo upraviť súčasné. Každý test musí byť pridaný alebo upravený

ešte upravovaním zdrojového kódu aplikácie. Toto umožní vývojárovi sa sústrediť na požiadavky ešte pred tým, ako píše program.

Spustenie testu a overenie či novo pridaný test nezlyhá. V prípade ak test prejde, daná časť je už v aplikácii implementovaná, preto netreba zdrojový kód meniť. Test môže aj zlyhať už z dopredu určených príčin. Vďaka tomu sa zvyšuje dôvera vývojára v metodológiu TDD.

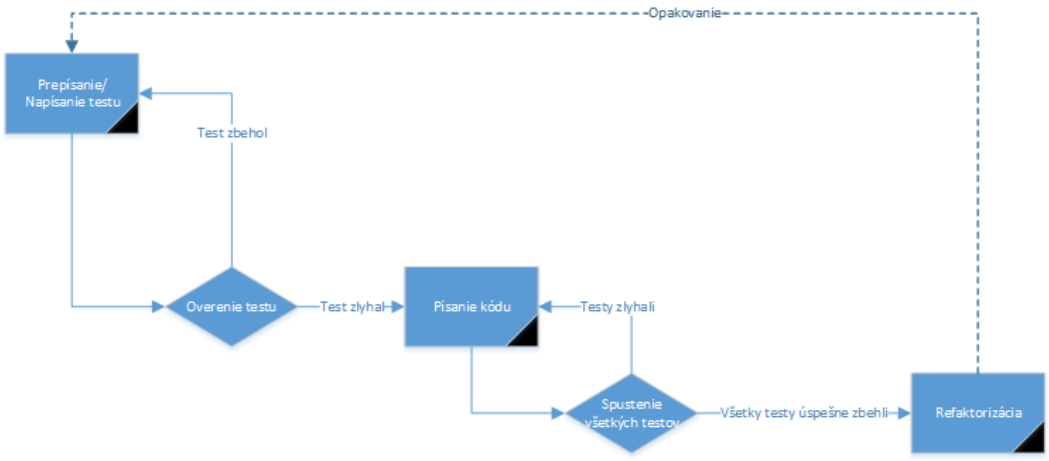
Písanie kódu je potrebné až dovtedy, pokiaľ nový test neprejde.

Spustenie všetkých testov: v prípade ak všetky testy prejdú, dá sa považovať kód za prijateľný. V prípade zlyhania niektorého z testov je potrebné upravovať kód, až kým všetky nie sú považované za akceptované.

Refaktorizácia: aj keď nový test prejde, je potrebné refaktorizovať kód. Tento proces zahŕňa odstránenie duplikátov, ktoré mohli vzniknúť, presunutie tried na ich logické miesta, premenovanie premenných na názvy, ktoré ich lepšie popisujú a ďalšie refaktorizačné procesy. Refaktorizácia výrazne zvyšuje udržateľnosť celého softwaru počas jeho životného cyklu.

Opakovanie celého cyklu, až pokiaľ testovacie prípady nie sú považované za prijateľné.

Predchádzajúce kroky sú grafický znázornené na obrázku 2.1



Obr. 2.1: Proces testovaia pomocou metodológie TDD

Najlepšie výsledky pri testovaní sa dajú dosiahnuť pri dodržaní štyroch zásad

1. *Nastavenie testov* a pripravenie systému na otestovanie.

2. *Vykonanie testov* a oddychytenie výsledkov ako sú výsledky funkcií alebo výstupné parametre. Tento krok musí byť ľahko vykonateľný a rýchly, lebo sa mnohokrát opakuje.

3. *Validácia testov* - overenie výsledkov testov a kontrola správnosti.

4. *Obnovenie testovaných objektov* a ich pripravenie na ďalšie testovanie tak, aby predchádzajúce testovanie neovplyvnilo aktuálne. Táto akcia dovoľuje spustenie viacerých testov po sebe.

2.3 Podobné existujúce systémy

V tejto časti spomenieme riešenia, ktoré sú podobné našej práci. Uvedieme ich výhody, nevýhody alebo popíšeme časti aplikácií, z ktorých sme čerpali inšpiráciu.

2.3.1 L.I.S.T.

L.I.S.T. [Jur13] vznikol ako bakalárska práca v roku 2013 na Univerzite Komenského. V súčasnej dobe je využívaný ako systém na vytváranie, odovzdávanie zadaní a na niektorých predmetoch aj ako ich automatické hodnotenie. V našej práci z neho využijeme ideu zobrazovania zadaní a prihlasovania na predmet. Inšpirovali sme sa i používateľským rozhraním hlavne čo sa týka rozloženia prvkov na jednotlivých podstránkach.

2.3.2 Google Docs[SH09]

Google docs sa skladajú z viacerých častí ako je napríklad textový editor, tabuľkový procesor, prezentačný software a mnoho iného. Všetky tieto technológie umožňujú online spoluprácu na jednom dokumente viacerým ľuďom. V našej práci nás však bude zaujímať iba textový editor. Tento textový editor bol hlavnou inšpiráciou na tvorbu práce, pretože umožňuje online editáciu textu. Editor sme si prispôsobili na potreby programovania, teda sme z neho odstránili takmer všetky možnosti formátovania textu, ktoré sú pri programovaní nepotrebné a pridali novú funkcionalitu

- **syntax highlighting** je funkcionalita, ktorá umožňuje farebné rozlišovanie zdrojového kódu a zároveň aj jednoduché zisťovanie syntaktických chýb

- **autocomplete** nám umožňuje jednoduché doplňanie textu pomocou klávesových skratiek ako napríklad premenné, metódy alebo rezervované slová

2.3.3 Stránka predmetov F.Gyárfáša

Stránka predmetov F. Gyárfáša slúži podobne ako L.I.S.T. na odovzdávanie zadaní v programovacom jazyku C++. Sú dve možnosti ako odovzdať riešenie: buď sa nahrá riešenie ako textový súbor, alebo je možnosť priameho písania kódu do editoru. Druhá možnosť nie je veľmi využívaná, lebo mnoho ľudí ju považuje za nepraktickú. Z tejto aplikácie je prebraná samotná idea testovania, možnosť spájať viacero ľudí do jedného zadania, ich spoločné hodnotenie a samozrejme bezpečné spúšťanie kódu na serveri.

2.3.4 GitHub

GitHub je internetová služba založená na hostovaní verzionovacieho systému *git*. Narozdiel od *git*-u, ktorý je prístupný len ako konzolová aplikácia, *GitHub* má používateľské webové rozhranie. V našom systéme nebudeme používať priamo *GitHub* a všetky jeho možnosti, ale preberieme si len časť, ktorá umožňuje porovnávanie súborov v grafickom rozhraní ako je ilustrované na obrázku 2.2. V našom systéme si užívatelia budú môcť porovnať dve rôzne verzie, ktoré odoslali na otestovanie.

Obr. 2.2: Porovnanie dvoch verzií súborov v systéme GitHub

2.3.5 Zhodnotenie

Ako sme spomenuli v predchádzajúcich častiach, náš systém bude pokrývať štyri rôzne služby. Z každého vymenovaného projektu si zoberieme to dôležité, ktoré je náplňou tejto práce. Odstránime nedostatky, ktoré tieto projekty majú a pridáme funkcionalitu tak, aby pokrývala všetky požiadavky na našu prácu

- **L.I.S.T.** inšpirácia pre správu predmetov, zadaní a rozloženie elementov na stránke
- **Google docs** online spolupráca na dokumentoch
- **Stránka predmetov F.Gyárfáša** spúšťanie zdrojového kódu na serveri, jeho hodnotenie a využitie metodológie Test driven development
- **GitHub** verzionovanie súborov projektu na serveri a ich porovnávanie

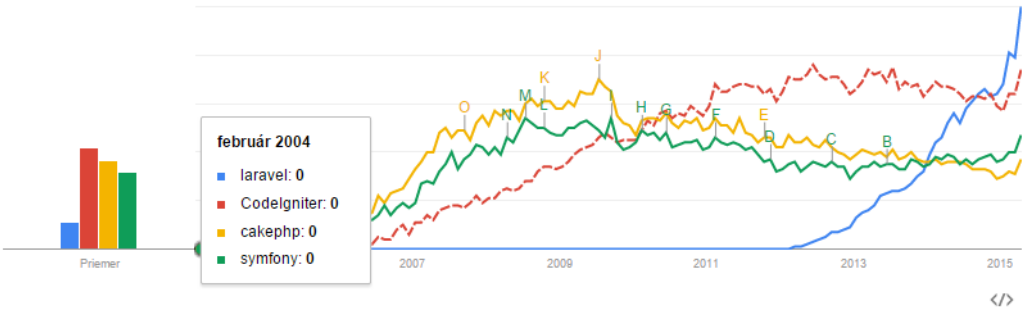
2.4 Použité technológie

Na vytvorenie aplikácie sme použili nástroje a technológie, ktoré sú v súčasnej dobe bežné pre vývoj webových aplikácií a dajú sa považovať za určitý štandard. Medzi ne patrí napríklad značovací jazyk *HTML5* alebo kaskádové štýly *CSS3*. Zo skupiny serverových jazykov sme využili *PHP5* a *node.js*. Na databázovú časť sme využili relačnú databázu *MySQL* a no-sql dátovú štruktúru *redis*. V nasledujúcej časti nebudeme popisovať tieto základné jazyky a technológie, ale rozoberieme si frameworky a využité nástroje.

2.4.1 Laravel

Celá aplikácia je postavená na PHP frameworku *Laravel* vo verzii 4 [Sau14]. V dobe písania práce framework existuje už vo verzii 5, ale počas implementácie bola táto verzia len vo fáze beta testovania, tak sme ju nevyužili. Pre tento framework sme sa rozhodli z dôvovdu, že v súčasnosti patrí medzi spičku medzi PHP frameworkami ako aj dokazuje obrázok 2.3, ktorý je prebraný zo štatistiky vyhľadávania výrazov v internetovom vyhľadávači Google. Ako je z grafu vidieť patrí medzi najmladšie frameworky, ale svoju popularitu si získal veľmi rýchlo a to pravdepodobne vďaka svojej elegantnej

syntaxi, ideálnemu oddeleniu logických častí aplikácie, vynikajúcej dokumentácii a pokročilému ORM.



Obr. 2.3: Porovnanie záujmu o PHP frameworky v čase

Tak ako mnoho iných webových frameworkov aj *Laravel* je postavený na architektúre MVC, ktorá oddeľuje dátové modely (databázu), spracovanie dát a ich následné vykreslenie užívateľovi.

- **Model** je terminológii frameworkov namapovaná entita z databázy do objektov. V Laraveli sa pre model využívajú objekty oddedené z triedy *Eloquent*. Vďaka tomu sa s databázou pracuje jednoduchšie, nie je potrebné písať *SQL* dotazy. Vďaka pokročilému ORM je možné vytvoriť vlastné objekty pre každú entitu v databáze, ktoré nám zapúzdria metódy a zjednodušia prácu s ňou.

- **View** je jednoducho povedané, časť aplikácie, ktorú užívateľ vidí a interaguje s ňou. Laravel využíva tzv. *Blade* pre kreslenie viewov. Blade je veľmi podobný HTML, ale umožňuje pokročilé funkcie ako je dedenie (rozširovanie) zobrazovaných častí, logické operácie vo viewoch, ale aj bezpečné vypisovanie dát od užívateľa.

- **Controller** je časť aplikácie, ktorá sa stará o spojenie modelov s vykreslením stránok (nemusí byť pravidlo, lebo pri vytváraní moderných *RESTful* aplikácií sa komunikuje napríklad len pomocou dát, zvyčaje vo forme *JSON*) a samozrejme o celú aplikačnú logiku.

2.4.2 Twitter Bootstrap

Twitter Bootstrap je zoskupenie viacerých elementov, ktoré zabezpečejú správne vykreslenie webovej stránky v prehliadači užívateľa. V súčasnej dobe je masívne využívaný na tisícoch webových stránkach na svete. Samotný framework je zložený z dvoch dôležitých častí

- **CSS** vzhľadom na to, že framework je na strane klienta, tak obsahuje kaskádové štýly. V našej práci sme sa rozhodli použiť *Twitter Boostrap 3*, ktorý na rozdiel od svojho predchodcu obsahuje aj podporu pre mobilné zariadenia. Základný stavebný kameň kaskádových štýlov frameworku je rozdelenie stránky na dvanásť stĺpcov, ktoré sa dajú ďalej deliť. Toto nám zabezpečí jednotný výzor na všetkých podstránkach a jednoduchšie štýlovanie celej aplikácie.

- **JavaScript a jQuery** je priamo implementované vo frameworku. Uľahčuje prácu programátorom, lebo sa v ňom nachádza mnoho funkcií, ktoré sa vyskytujú na mnohých webových stránkach ako je napríklad použitie tooltipov, modálnych okien, dropdownových menu a iné.

Vzhľadom na to, že framework je rozšírený po celom svete, tak na neho vznikajú aj rôzne rozšírenia. My v aplikácii budeme používať WYSIWYG editor *Summernote* a zvýrazňovač zdrojového kódu *google-code-prettify*.

2.4.3 node.js a jeho rozšírenie ShareJS

Node.js je javascriptový server-side framework, ktorý poskytuje event-driven architektúru a asynchrónne I/O (Input/Output). Keďže naša aplikácia je založená na tom, že potrebujeme online synchronizáciu pre všetkých užívateľov, tak *node.js* bol jasná voľba pre túto časť aplikácie. Avšak samotný *node.js* nemá túto funkciu priamo v sebe zabudovanú, preto je potrebné použiť modul *ShareJS*. *ShareJS* je modul, ktorý umožňuje komunikáciu server-klient. Toto je kľúčové vzhľadom na to, že potrebujeme mať prehľad o aktuálne pripojených klientoch na stránke a umožňovať im komunikáciu v reálnom čase.

2.4.4 redis

Ako sme spomenuli v predchádzajucej kapitole, tak na živú komunikáciu klientov využívame *ShareJS*. Keďže týchto operácií môže byť veľmi veľa naraz (napríklad každé jedno stlačenie klávesy v online editore) je potrebné mať dátovú štruktúru, ktorá bude dostatočne rýchla. Spočiatku bola možnosť použiť *MySQL* databázu na synchronizovanie, ale to sa ukázalo ako neefektívne, pretože bola spomalená zvyšná časť aplikácie, ktorá komunikovala s *MySQL*. Nielen preto sme sa rozhodli použiť na synchronizáciu *redis*, ale zároveň aj preto, lebo je odorúčaný ako dátové úložisko pre *ShareJS*. *Redis* si uchováva svoje dáta v mapovej štruktúre, preto ku každej live zmene (editoru, chatu. . .) pristupujeme ako ku kľúču v pamäti. Keďže *redis* podporuje priamo optimálnu durabilitu (schopnosť držať si informácie o kľúči a jeho hodnote kvôli zvýšeniu rýchlosti), je vhodný na našu aplikáciu.

2.4.5 Ace

Ace je moderný javascriptový editor, ktorý umožňuje tvorbu zdrojového kódu priamo na stránke prehliadača. Editor sme si zvolili z dvoch dôvodov

- editor je v súčasnej dobe aktívne vyvíjaný a patrí medzi špičku voľne dostupných editorov a je využívaný mnohými veľkými spoločnosťami na svete
- editor je priamo podporovaný knižnicou *ShareJS*, čo výrazne zjednoduší programovanie tohoto komponentu do aplikácie

2.4.6 Summernote

Summernote je webový textový WYSIWYG editor, ktorý umožňuje tvorbu HTML obsahu bez toho aby používateľ potreboval vedieť tento značkovací jazyk. Tento editor sme si zvolili z dôvodu, že je priamo postavený pre platformu *Twitter Bootstrap* a preto ho nie je potrebné špeciálne prispôsobovať HTML kód, ktorý nám vyprodukuje.

2.4.7 g++ kompilátor

Keďže je potrebné každé jedno odovzdané riešenie otestovať, tak treba vždy vytvoriť spustiteľný súbor. Pre *g++* sme sa rozhodli pretože je skoro v každej linuxovej

distribúcii priamo predinštalovaný a zároveň umožňuje aj kompiláciu z príkazového riadku. Práve táto kompilácia z príkazového riadku je pre nás kľúčová, lebo ju budeme spúšťať z PHP skriptov, pre ktoré je nemožné a neefektívne spustiť grafickú aplikáciu na pozadí.

2.4.8 Google C++ Testing Framework

Pre otestovanie riešenia sú potrebné testovacie funkcie, ktoré nám zabezpečia testovanie. Preto sme sa rozhodli siahnuť po určitom frameworku, ktorý je priamo prispôsobený pre unit testovanie. *Google C++ Testing Framework* je pre nás optimálny z viacerých hladísk

- ako je naznačené v predchádzajúcej časti, tak pre nás je kľúčová kompilácia z príkazového riadku. *Google C++ Testing Framework* nám to dovoľuje, čo je jeho veľkým plus, lebo nepotrebujeme grafické rozhranie
- po skončení testovania, nám framework vráti výsledky o testovaní v podobe XML súborov, ktoré sú už ľahké pre spracovanie v PHP skriptoch
- keďže potrebujeme mať otestovanie kódu veľmi rýchle (užívateľ nemá rád, keď musí čakať), tak framework nám to umožňuje z viacerých hladísk. Samotná tvorba testovacích súborov nie je komplikovaná a teda prebehne rýchlo a zároveň aj framework je optimalizovaný pre rýchlosť, preto samotná testovacia časť netrvá dlho

Kapitola 3

Návrh riešenia

Táto kapitola sa venuje návrhu riešenia, podľa ktorého je aplikácia implementovaná. Kapitola má priblížiť čitateľovi aplikáciu z vonkajšieho (užívateľského) pohľadu. Obsahuje popis používateľských skupín, požiadavky na systém a detaily o niektorých dôležitých častiach systému.

3.1 Funkčné požiadavky

Cieľom tejto práce je vytvoriť informačný systém na tvorbu zadaní a unit testov v programovacom jazyku C++. Keďže takýchto implementácii existuje mnoho, tak aplikácia bude rozšírená o možnosť riešenia, testovania a bodovania programovacích úloh online pre viacero ľudí naraz.

3.1.1 Požadovaná funkcionalita aplikácie

Skôr než si uvedieme a popíšeme možnosti, ktoré bude aplikácia umožňovať, tak si zavedieme slovník pojmov, ktoré budeme v nasledujúcej kapitole používať.

3.1.1.1 Zadanie

Časť aplikácie, ktorá obsahuje slovný popis zadania a pridružené entity, ktoré potrebuje užívateľ pri riešení úlohy. Skladá sa z troch častí

- **text** zadania obsahuje slovný popis zadania, ktorý bližšie špecifikuje zadanie
- **predefinované súbory** k zadaniu sú základné súbory potrebné ku spusteniu testov. Obsahujú predefinované triedy, metódy, štruktúru projektu, ktorú je po-

trebné dodržať. Užívatelia, ktorí riešia zadanie ich dopĺňajú alebo upravujú až kým nevyriešia zadanie

- **testovacie prípady** sú potrebné k overeniu správnosti zadania. Každý testovací príklad sa skladá z blokov, sekcií a samotného testu

3.1.1.2 Riešenie

Za riešenie sa dá považovať skompilovateľný zdrojový kód, ktorý vytvorí užívateľ (skupina užívateľov) a je akceptovaný všetkými testovacími prípadmi.

3.1.1.3 Skupina

Skupina je množina užívateľov (môže byť aj jednoprvková), ktorá spolupracuje na riešení. Pre každé zadanie je možnosť vytvorenia vlastnej skupiny. Členovia skupiny vidia na stránke riešenia svoje spoločné riešenie v online editoroch, skupinový chat a vlastné testovacie prípady.

3.2 Používatelia systému

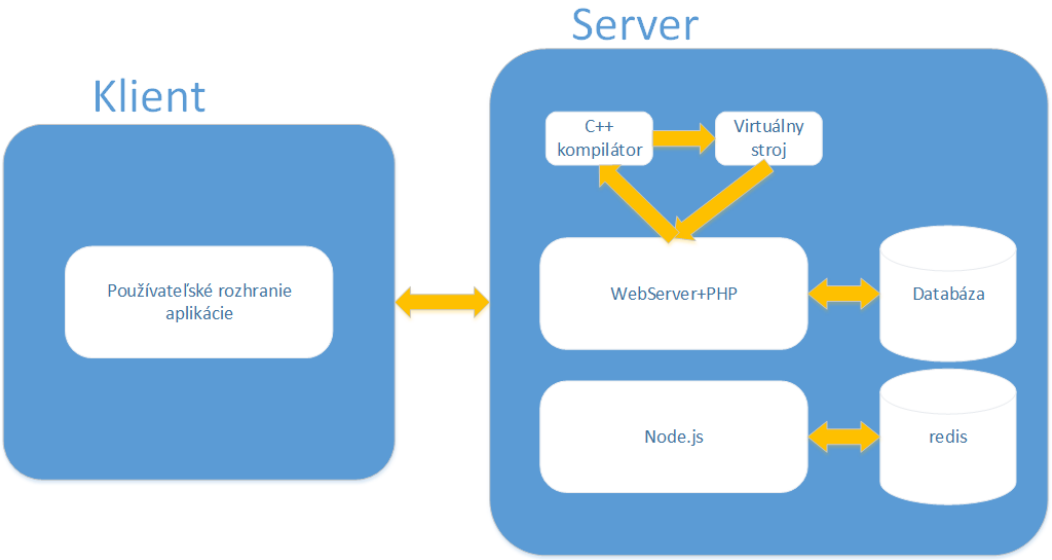
Aplikácia bude bude obsahovať tri rôzne skupiny používateľov

- **administrátor** bude mať na starosti správu celej aplikácie. Bude môcť spravovať ostatných užívateľov, predmety
- **učitelia** budú mať právo spravovať predmet, ku ktorému sú priradení, pridávať žiakov do predmetu, vytvárať a upravovať zadania
- **žiaci** budú mať právo prezerať si zadania, riešiť ich, vytvárať si vlastné unit testy a spravovať si skupiny ku každému zadaniu

3.3 Architektúra aplikácie

Keďže výsledkom tejto práce má byť webová aplikácia, postavíme ju na štandardnej architektúre klient-server. Za klientskú časť aplikácie sa v jednoduchosti dá považovať všetko, čo užívateľ vidí vo svojom internetovom prehliadači a za server všetku logiku,

ktorá sa deje na počítači, na ktorom je aplikácia spustená. Grafické znázornenie architektúry je možné vidieť na obrázku 3.1.



Obr. 3.1: Architektúra aplikácie

3.4 Bezpečnostné požiadavky na systém

Webová aplikácia je dostupná prakticky z ktoréhokoľvek miesta na svete, ktoré je pripojené na internet. Práve pre toto je potrebné aplikáciu zabezpečiť voči nežiadúcim vplyvom. Za zabezpečenie sa však nepovažuje zabezpečnenie aplikácie ako takej, ale aj iných aplikácií a samotného počítača, na ktorej je naša aplikácia nainštalovaná a používaná. Medzi bezpečnostné opatrenia mimo aplikácie patria hlavne

- **zabezpečnenie operačného systému** - zodpovedá za ňu hlavne administrátor servera. Medzi hlavné požiadavky na zabezpečenie operačného systému patrí inštalácia bezpečnostných záplat, použitie firewallu, ale aj správne a rozumné použitie užívateľských práv (aby iná aplikácia alebo škodlivý kód nemohol zasahovať do súborov projektu)
- **zabezpečnie dát a ich integrita** - hlavný dôraz by sa mal klásť na neoprávnené a neodborné zásahy do dát aplikácie, ale aj na zálohovanie dát na externé pamäťové úložisko v prípade poruchy serveru

- **zabezpečenie fyzických častí serveru a pridružených častí siete** - jedná sa o fyzické zabezpečnie serveru a dôležiých zariadení (sieťová architektúra, záložný zdroj napájania...) v prípade nepredvídaných udalostí, ktoré by mohli ohroziť beh aplikácie

Vyššie spomenuté požiadavky však nedokážeme ovplyvniť našou implementáciou. Preto budeme dodržiavať hlavne požiadavky, ktoré naša implementácia aplikácie môže priamo ovplyvniť na počítači, na ktorom je spustená. Medzi bezpečnostné požiadavky na aplikáciu patria

- **overenie oprávnených osôb** - je dôležité postaviť aplikáciu tak, aby každá používateľská skupina mala jasne vymedzené práva a dali sa ľahko pridávať alebo odoberať každému užívateľovi

- **zabezpečenie používateľských vstupov** - aplikácia má natívne zamedziť vplyvom škodlivých používateľských vstupov a pokusov o napadnutie a zničenie systému pomocou týchto vstupov. Hlavný dôraz sa kladie na zabezpečenie SQL injection (pokus o napadnutie databázy pomocou škodlivých databázových dotazov) a XSS útokov (pokus o spustenie škodlivého, väčšinou javascriptového, kódu, ktorý môže škodiť v rozsahu od rozbitia dizajnu stránky až po získanie citlivých údajov uložených v databáze), ktoré sú dobre známe a zároveň aj často využívané útočníkmi v súčasnej dobe

- **zabezpečnie voči škodlivému kódu spusteného z aplikácie** - vďaka možnosti spúšťať vlastný kód na serveri môže naša aplikácia ponúknuť útočníkovi dvierka do počítača, na ktorom je aplikácia spustená, a tým získava takmer neobmedzné možnosti napadnutia systému. Preto je dôležité mať riadne zabezpečenú túto časť aplikácie, aby nebola ohrozená nielen naša aplikácia, ale aj iné aplikácie a hlavne samotný operačný systém

3.5 Grafický návrh aplikácie

V tejto časti sa budeme venovať grafickému návrhu aplikácie. Celý systém bude riešený pomocou frameworku *Twitter Bootstrap*. Vďaka tomuto nástroju sa nám podarí rýchlo

vytvoriť požadovaný výzor stránky, ktorý bude intuitívny, ľahko ovládateľný a dostupný pre rôzne druhy zariadení. Aplikácia sa bude skladať z viacerých podstránok, z ktorých najdôležitejšie vyberieme a popíšeme.

- **základný funkčný layout** je znázornený na obrázku 3.2

Obr. 3.2: Základný výzor aplikácie bez aktívnej obrazovky

- **editor** je základná časť aplikácie a je zobrazený na obrázku 3.3. Editor by mal hlavne poskytovať jednoduchý syntax highlight, dopĺňanie textu a zobrazovanie čísiel riadkov

- **chat** by mal byť dostupný pri zobrazení riešenia a mal by byť dostupný len skupine. Bude sa nachádzať na pravej strane obrazovky a bude upozorňovať užívateľa na novú správu, prihlásenie, odhlásenie iných užívateľov. Samozrejmosťou bude aj farebná notifikácia a možnosť skrytia editoru tak, aby neprekážal počas programovania. Približný návrh chatu je možné vidieť na obrázku 3.4

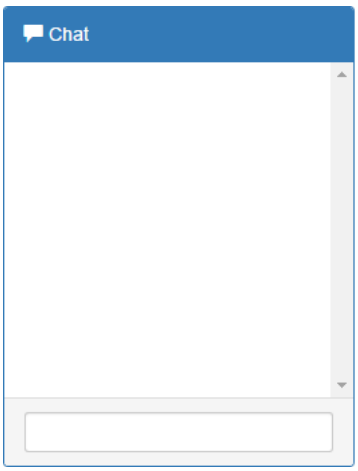
```cpp
#include <string>

using namespace std;

const double NEKONECNO = 9999.9999;
const int POCET_RADOV = 9;
const int POCET_PISMEN = 3;
const char cifry[POCET_RADOV][POCET_PISMEN] = {
{'I','V','X'}, {'X','L','C'}, {'C','D','M'},
{'M','P','Q'}, {'Q','R','S'}, {'S','T','U'},
{'U','W','Y'}, {'Y','Z','E'}, {'E','F','G'}
};
const char NULA = 'O';
const char MINUS = '-';
const int POCET_OPERATOROV = 9;
const char operatory[POCET_OPERATOROV] = {'+','-','*','/','&','|','>','<','='};

class RIMSKA_KALKULACKA {
  bool nachadzaSaVMnozine(char c, int mnozina);
  bool rimskePismeno(char c);
  bool povolenyOperator(char c);
  double vypocetSkupiny(string skupina, int uroven);
 public:
//1.uloha - prevodnik z rimskych na arabske (zly vstup vracia NEKONECNO)
  double konverziaRimskychNaArabske(string rimskeCislo);
//2.uloha - prevodnik z arabskych na rimske
  string konverziaArabskychNaRimske(double cislo);
//3.uloha - kalkulacka s argumentmi +, -, *, /
  double kalkulackaArabska(char oper, double op1, double op2);
//4.uloha - kalkulacka rimska
  string kalkulackaRimska(string vyraz);
};
const bool DUMMY_BOOL = false;
const int DUMMY_INT = 0;
const string DUMMY_STRING = "";
```

Obr. 3.3: Základný výzor editoru aj s kusom zdrojového kódu



Obr. 3.4: Návrh chatu pre skupinu

Kapitola 4

Implementácia riešenia

V tejto kapitole sa budeme venovať samotnej implementácii systému podľa návrhu. Cieľom kapitoly je priblížiť čitateľovi aplikáciu z vnútorného pohľadu.

4.1 Databáza

Ako pri každej webovej aplikácii aj my budeme využívať databázu. Avšak v aplikácii budú použité dve databázy kvôli zrýchleniu systému a oddelenie synchrónnych a asynchrónnych častí. Nesynchrónna časť napríklad užívatelia, testy, zadania a podobne budú uložené v *MySQL*. Synchrónna časť bude využívať *redis* kvôli rýchlosti a jednoduchšej implementácii.

4.1.1 MySQL

V tejto časti budeme popisovať *MySQL* databázu, jej tabuľky a ich význam v systéme.

4.1.1.1 Popis tabuliek a entitno-relačný diagram databázy

Každá spomenutá tabuľka implicitne obsahuje stĺpec *id*, čo je primárny kľúč v každej tabuľke a stĺpce *created_at*, *updated_at*, ktoré sú frameworkové a držia časové informácie o vytvorení záznamu a jeho poslednej zmene.

blocks Obsahuje testovacie bloky ku každému zadaniu.

files Do tejto tabuľky sa zapisuje vždy, keď sa spustí zadanie na otestovanie. Vždy pri zapisovaní sa zvýši verzia súborov uložených pre skupinu.

groupmembers Tabuľka obsahuje referencie na užívateľov a skupiny. Obsahuje napárovanie užívateľa k skupine.

groups V tabuľke sa nachádzajú informácie o každej skupine, jej zakladateľovi, stave a zadaní, ku ktorému je priradená.

migrations Základná tabuľka frameworku, ktorá slúži na synchronizovanie zmien v databáze ako je napríklad zmena názvu tabuľky, editácia stĺpcov, upravovanie kľúčov a iné.

owntests V tabuľke sa nachádzajú informácie o vlastných testoch, ktoré si skupina vytvorí. Obsahuje stĺpce kód pred testom, testovaná funkcia, porovnanie, očakávaná hodnota, kód po teste a referenciu na skupinu.

participants Tabuľka, ktorá je dôležitá na identifikovanie osôb, ktoré sú prihlásené na predmet. Obsahuje referencie na predmet a na osobu, ktorá sa naň prihlásila.

password_reminders Tabuľka, do ktorej sa zapisujú údaje potom ako užívateľ zažiada o zmenu hesla. Obsahuje e-mail užívateľa, ktorý o zmenu zažiadal a náhodne vygenerovaný unikátny token, ktorý užívateľ použije pri zmene hesla.

revisions Tabuľka vytvorená komponentom *VentureCraft/revisionable*, ktorá slúži na zaznamenávanie zmien v dátach. Ukladá si tabuľku, číslo primárneho kľúču, stĺpec, ktorý bol zmenený, užívateľa, ktorý zmenil dáta, starú a novú hodnotu zmeneného riadku.

solutions Obsahuje súbory, ktoré sú unikátne pre každé riešenie, skupinu, ku ktorej sú priradené a zároveň si aj drží dôležitú informáciu o náhodnom tokene pre každý súbor, ktorý sa používa ako kľúč v *redis* úložisku pre pripojenie súborov na synchronizáciu.

subjects Tabuľka, v ktorej sú uložené všetky predmety. Obsahuje názov predmetu, učiteľa, rok a semester, v ktorom sa predmet bude vyučovať.

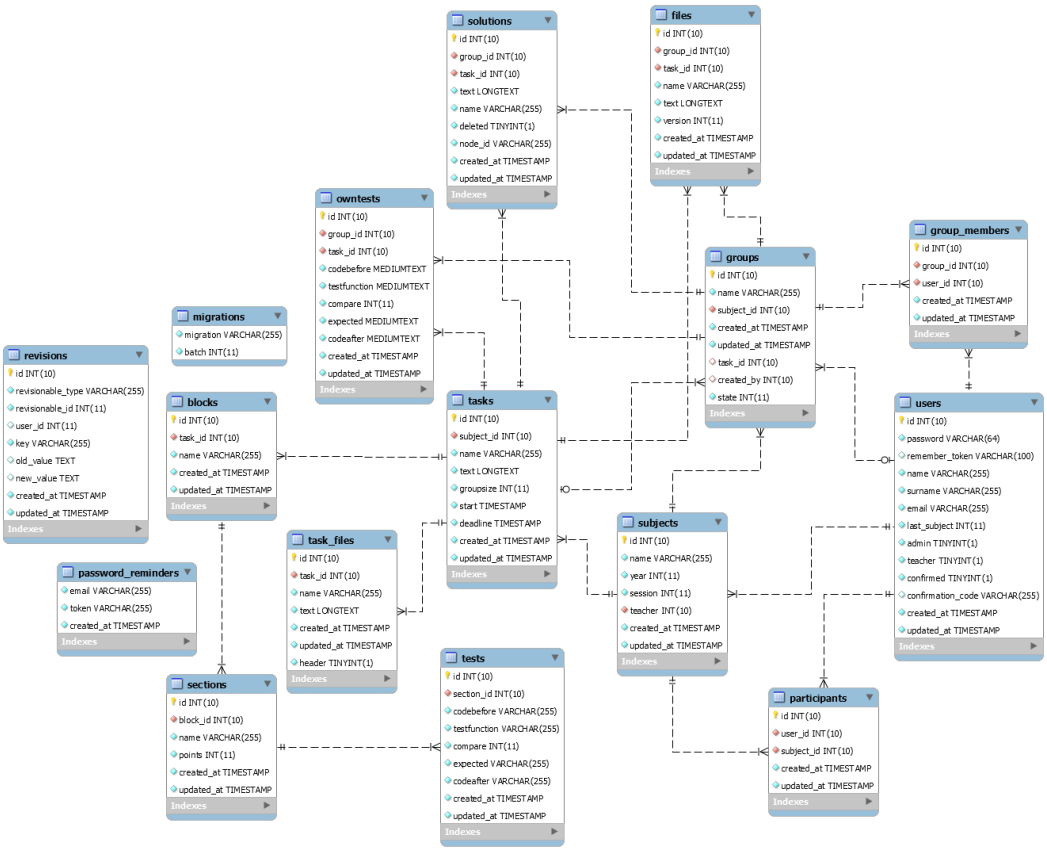
task_files Dôležitá tabuľka, ktorá sa napĺňa pri vytvorení zadania s preddefinovanými súbormi na riešenie, ich názvami a príznakom, či sa jedná o hlavičkový súbor.

tasks V tabuľke sa nachádzajú informácie o každom jednom vytvorenom zadaní. Obsahuje stĺpce ako sú meno zadania, predmet, maximálna veľkosť skupiny pre zadanie, začiatok a koniec zadania.

tests Tabuľka je takmer identická s tabuľkou *owntests*, len s rozdielom, že tieto testy majú cudzí kľúč na zadanie, a teda sú verejné pre všetkých a nie len pre skupinu.

users Obsahuje informácie o všetkých registrovaných užívateľov v systéme. Štruktúra vychádza z predefinovanej tabuľky, ktorú poskytuje framework Laravel.

Vyššie spomenuté tabuľky sú poprepájané pomocou referencií, ktoré sú vidieť na nasledujúcom obrázku 4.1



Obr. 4.1: Entitno-relačný diagram databázy

4.1.2 redis

Sekcia obsahuje informácie o využití *redis* v našom projekte.

4.1.2.1 Popis kľúčov

Keďže redis je odlišný typ úložiska od *MySQL*, tak nemá zmysel popisovať jeho tabuľky, ale len typy kľúčov, ktoré sa používajú. V ďalšej časti popíšeme prefixy kľúčov, ktoré používame.

ShareJS:ops:code:* Tento kľúč slúži na synchronizovanie a ukladannie každého jedného súboru, s ktorým sa pracuje. Posledný parameter je unikátny token, ktorý je uložený v *MySQL* databáze v tabuľke *solutions* v stĺpci *node_id*.

ShareJS:ops:groups Kvôli lepšiemu prehľadu pri správe skupín sa využíva tento kľúč na notifikáciu všetkých užívateľov, ktorí sú prihláseni v časti úprava skupiny, o rôznych zmenách v skupine ako je napríklad pridanie člena, odchod člena zo skupiny, alebo aj informácie o samotnom vytvorení a odstránení skupiny. Vzhľadom na to, že nie je potrebné odlišovať užívateľov, ktorí prijímajú správu z tohoto kľúča, tak sa nevyužíva žiadny unikátny kľúč, ale ako správa sa pošle unikátne číslo zadania, v ktorom sa zmenili niektoré dáta.

ShareJS:ops:shout:* Pomocou kľúča sa spája chat celej skupiny. Posledný parameter je zreťazené *id* zadania a *id* skupiny.

ShareJS:doc:tests:* Kľúč slúži na spojenie skupiny pri vytváraní vlastných testov. Unikátne *id* sa skladá zo zreťazeného *id* zadania a *id* skupiny.

ShareJS:ops:toggle:* V prípade ak jeden člen skupiny spustí testovanie zadania, tak pomocou tohoto kľúča sa rozpošle správa celej skupine o tom, že bolo spustené testovanie a všetkým členom sa zablokuje editor až do skončenia testovania. Ako unikátny parameter sa využíva zreťazené *id* zadania a *id* skupiny.

4.2 Editor a ostatné synchrónne časti

V tejto časti si rozoberieme funkcionalitu komponentu editor, ktorý umožňuje online synchronizáciu zdrojového kódu.

4.2.1 Ace

Na klientskú časť, teda zobrazenie zdrojového kódu riešenia zadania sa využíva javascriptový editor *Ace* (Ajax.org Cloud9 Editor) v minulosti známy ako *Bespin* alebo *Skywriter*. Editor v našej aplikácii budeme pripájať na server pomocou nasledujúceho kódu.

```
1 sharejs.open("code:" + node_id, 'text', channel,
2   function (error, doc) {
3           docs[node_id] = doc;
4           docs[node_id].attach_ace(editors[node_id]);
5 });
```

Táto časť kódu inicializuje editor pre online použitie. Podobne sa pripája aj na ďalšie časti, ktoré potrebujeme synchronizovať.

4.2.2 ShareJS

Na serverovú časť editora sa využíva *node.js* a jeho rozšírenie *ShareJS*. Táto časť servera beží na samostatnom porte a vždy po upravení textu sa naň pošle požiadavka napríklad v tvare *insert:'hi', position:50*. Server túto požiadavku spracuje a pošle všetkým prihláseným klientom, počúvajúcich na určitom kanáli, správu. V tejto správe je zahrnutá aktuálna zmena a všetci pripojení klienti na túto požiadavku zareagujú a vykonajú určitú akciu.

Ako bolo naznačené v predchádzajúcej kapitole, tak tento komponent sa nevyužíva len na synchronizáciu textu, ale aj ostatné asynchrónne časti.

- **chat -** každá skupina má svoj vlastný chat, ktorý vidia len členovia v skupine. Vďaka tomu, že *ShareJS* môže posielať správy v tvare *JSON*u je ľahké prerobiť túto knižnicu na to, aby sa dala využiť ako chat. Pri odoslaní správy sa na kanál chatu konkrétnej skupiny pošle JSON v tvare *'user_id':id_užívateľa,*

'text':'správa'. Podobne ako *Ace* editor, tak aj chat sa pripája na server pomocou príkazu, ktorý je možné vidieť v skrátenej forme ďalej.

```
1 sharejs.open("shout:" + docName, 'text', channel, function (error
    , doc) {
2   doc.on('shout', function (msg) {
3         addShout(msg);
4     });
5
6     doc.shout(msg);
7 });
```

Na posielanie správ sa používa metóda *shout(msg);*, ktorá ako parameter dostáva správu v tvare *JSON*u a na prijímanie správ spúšťač *on('shout', function (msg) { })*.

- **správa vlastných testov -** každá skupina si môže vytvárať vlastné testovacie scenáre, ktoré vidia len členovia tejto skupiny. V prípade ak sa nejaký test zmení, pridá alebo odstráni, tak všetci členovia sú o tom hneď informovaní.
- **správa skupín -** kvôli väčšiemu používateľskému komfortu sa tento komponent využíva aj v časti správa skupín, kde je tiež využitý na real-time notifikáciu o zneme v skupinách.

4.3 Testovanie, Google C++ Testing Framework

Vzhľadom na to, že všetky testy musia byť spustené v konzolovom režime, sme sa rozhodli, že na testovanie sa bude využívať testovací framework *Google C++ Testing Framework*. V tejto časti si rozoberieme inštaláciu frameworku, vytváranie súborov pre framework, jeho použitie a získanie výsledkov.

4.3.1 Inštalácia

Sfunkčnenie frameworku sa delí na dve časti a tými sú buildovanie, kompilácia a inštalácia (zlinkovanie headerov) do systému.

4.3.1.1 Buildovanie, kompilácia

Ako prvý krok je potrebné získať kópiu frameworku do systému. Dá sa to s využitím príkazu alebo manuálnym získaním súborov z oficiálneho repozitára.

```
$ wget http://googletest.googlecode.com/files/gtest-1.7.0.zip
```

Po úspešnom získaní súborov si zdrojové súbory vybuildujeme pomocou príkazov

```
$ unzip gtest-1.7.0.zip
$ cd gtest-1.7.0
$ ./configure
$ make
```

4.3.1.2 Linkovanie headerov

Linkovanie headerov sa môže odlišovať vzhľadom na rôzne použité verzie Linux. My využívame *Ubuntu*, tak budeme popisovať inštaláciu na tento systém. Skopírujeme headre a *.lib* súbory do systému pomocou príkazov

```
$ sudo cp -a include/gtest /usr/include
$ sudo cp -a lib/.libs/* /usr/lib/
```

Kvôli kontrole správnosti inštalácie je dobré si skontrolovať, či sa všetko naozaj nainštalovalo a to za pomoci príkazu

```
$ sudo ldconfig -v | grep gtest
```

V prípade ak dostaneme takýto alebo podobný výsledok (záleží od použitej verzie frameworku), tak inštalácia prebehla v poriadku.

```
libgtest.so.0 -> libgtest.so.0.0.0
libgtest_main.so.0 -> libgtest_main.so.0.0.0
```

4.3.2 Vytváranie súborov pre framework

Vytváranie súborov pre testovanie prebieha v dvoch krokoch. Tým prvým je vytvorenie testovacieho súboru z testov, ktoré sú zadefinované v databáze a tým druhým je vytvorenie súborov z práve odoslaného riešenia.

4.3.2.1 Vytvorenie testovacieho súboru

Žiadaný testovací súbor má tvar

```
#include <limits.h>
// nasleduju hlavickove subory
// riesenia v tvare #include "riesenie.h"
#include "gtest/gtest.h"

TEST(NAZOV_BLOKU, NAZOV_SEKCIE){
  EXPECT_EQ(volana_funkcia(), 'vysledok funkcie');
}
```

Testovací súbor sa vyskladá vždy pri každom spustení testovania a to pomocou nasledujúcej skupine PHP skriptov.

Najskôr si získame potrebné hlavičkové súbory, ktoré budeme includovať v testovacom súbore.

```
foreach ($taskFiles as $taskFile) {
  $fileNames[] = '#include "' . storage_path() . '/' .
  $task_id . $group_id . '/' .
  self::prepareString($taskFile->name) . '"';
}
$content .= implode(PHP_EOL, $fileNames) . PHP_EOL;
```

Získajú sa všetky sekcie a bloky z databázy, ktoré sú vytvorené pre testované zadanie.

```
foreach ($blocks as $block) {
  $sections = $block->sections()->get();
  foreach ($sections as $section) {
    $content .= "TEST(" . self::prepareString($block->name) .
    ", " . self::prepareString($section->name) . ") {" . PHP_EOL;
```

Pre vloženie testovacích prípadov stačí získať všetky testy, ktoré patria ku konkrétnej sekcii a to pomocou.

```
$tests = $section->tests()->get();
foreach ($tests as $test) {
  $content .= "  " . $test->codebefore . PHP_EOL;
  switch ($test->compare) {
    case Test::EQUAL:
```

```
6         $content .= "  EXPECT_EQ(";
7         break;
8       }
9     $content .= self::prepareString($test->expected) . ", " .
10    self::prepareString($test->testfunction) . ");" . PHP_EOL;
11    $content .= "  " . $test->codeafter . PHP_EOL;
12  }
```

V prípade ak má skupina zadeklarované vlastné testy, tak sa postupuje identicky ako pri získavaní testov, ktoré sú platné pre všetkých.

Nakoniec už máme celý súbor vyskladaný, tak nám ho stačí zapísať na disk pomocou funkcie z frameworku Laravel.

```
1 File::put(storage_path() . '/' . $task_id . $group_id . '/test.cpp',
    $content);
```

4.3.2.2 Vytvorenie súborov z riešenia

V predchádzajúcej kapitole sme si popísali ako sa vyskladá testovací súbor. Avšak pre správny chod je potrebné pripojiť k testovaniu aj testované súbory. Toto sa deje v troch krokoch

1. Keďže systém podporuje jednoduché verzionovanie riešení, tak v prvom kroku si zistíme aktuálne číslo verzie riešenia skupiny a to pomocou dotazu na databázu

```
1 DB::table('files')->where('group_id', '=', $input['group_id'])->
    max('version');
2 if ($version == NULL) {
3   $version = 0;
4 }
```

2. V druhom kroku si všetky súbory uložíme do databázy s navýšenou verziou. Súbory sa ukladajú vždy aj v prípade, že sa nedajú skompilovať alebo sa v nich nachádza viac chýb ako v predchádzajúcich verziách

3. V poslednom kroku je potrebné všetky tieto súbory uložiť na disk tak, aby boli čitateľné pre kompilátor a zároveň je aj potrebné zapamätať si v premenných cesty k týmto súborom, ktoré budú použité v testovacom súbore

Posledné dva kroky sa vykonávajú v nižšie popísanom cykle

```
1  foreach ($input['files'] as $file) {
2    $filedata = array(
3      'task_id' => $input['task_id'],
4      'group_id' => $input['group_id'],
5      'name' => $file['name'],
6      'text' => $file['text'],
7      'version' => ($version + 1),
8    );
9    (new SolutionFile($filedata))->save();
10   $includefiles .= $path . '/' . $file['name'] . ' ';
11   File::put($path . '/' . $file['name'], $file['text']);
12 }
```

4.4 Virtuálne prostredie

Virtuálne prostredied je časť aplikácie, ktorá umožňuje bez rizika spúšťať cudzí kód. Medzi hlavné riziká v našej aplikácii patria - zle odladený zdrojový kód (nekonečné while cykly, nekonečné rekurzie, ...), bezpečnostné riziká (pokus o zapisovanie do súborovej štruktúry operačného systému, vyvolávanie iných aplikácií a procesov cez zdrojový kód, ...).

Zjednodušene sa dá povedať, že sa jedná o operačný systém, ktorý beží pod hlavným operačným systémom. Tento vyvolaný virtuálny systém dostane vždy pri vytváraní vymedzené prostriedky v počítači, medzi ktoré patria

- **pamäť** v našom prípade každý virutálny stroj dostane 45 MB z pamäte RAM.
- **súborovú štruktúru** všetky virutálne stroje majú vlastnú adresárovú štruktúru, do ktorej môžu zapisovať, meniť a zároveň neohrozia chod celého systému ako aj ďalších strojov.
- **čas** všetky stroje bežia určitú dobu. V našej aplikácii je tento čas vymedzený na dobu 7 sekúnd, keďže sa nepredpokladá, že by testy mali trvať dlhšie ako túto dobu.

Vzhľadom na to, že nie je úplne triviálne tento virtuálny stroj sfunkčniť, tak v ďalšej kapitole si uvedieme základné kroky inštalácie a jeho spúšťanie z PHP skrip-

tov. Väčšina krokov v nasledujúcej kapitole je prebratých a poupravovaných pre naše potreby z bakalárskej práce Pavla Leščinského [Les14], kde sú vynikajúco rozpísané všetky teoretické poznatky.

4.4.1 Inštalácia

Inštalácia virtuálneho stroja je závislá od použitej verzie a hlavne od použitej linuxovej distribúcie. Preto by bolo takmer nemožné všeobecne popísať všetky kroky a prerekvizity, ktoré by boli pokrývali toto veľké množstvo distribúcií. My, na našom serveri, používame linuxovú distribúciu *Ubuntu*, ktorá je odvodená od distribúcie *Debian*. Preto je možné použiť príkaz *sudo apt-get install user-mode-linux*, ktorý nám zabezpečí to, že sa do operačného systému doinštalujú všetky potrebné prerekvizity pre spustenie virtuálneho stroju.

Pred spustením virtuálneho stroja je potrebné ho nakonfigurovať a pripraviť jeho systémové súbory. Keďže virtuálny stroj predstavuje rozšírenie jadra *Linuxu*, tak pre úspešný beh potrebuje rovnakú architektúru systému (napríklad počet bitov operačného systému) a podobnú verziu jadra ako má hostiteľský operačný systém. V našom prípade sa jednalo o jadro linuxu, ktoré sa nachádza v priečinku */boot* a to konkrétne súbor */boot/initrd.img-3.13.0-46-generic* (ako bolo spomenuté na začiatku kapitoly, tak verzie súborov môžu byť rôzne v závislosti od použitej verzie operačného systému). Pomocou nasledujúcich krokov sme vytvorili a pripravili zavádzacie súbory pre virtuálny stroj

1. Ako prvý krok je potrebné si vytvoriť nový priečinok, kam zavedieme súbory a to príkazom *mkdir initrd*. Otvoríme si tento priečinok pomocou príkazu v konzole *cd initrd*. Odporúčam si aj skopírovať zrojový súbor z priečinku */boot*, v prípade ak náhodou niektorá z operácii zlyhá.

2. V tomto kroku si inicializačný súbor rozbalíme do novovytvoreného priečinka. Toto je možné vďaka tomu, že zavádzací súbor je štandarný archív typu *cpio gzip*, ktorý rozbalíme pomocou príkazu *gzip -dc /boot/initrd.img-3.5.0-23-generic |cpio -id*. V tejto chvíli už máme celý vituálny stroj rozbalený v priečinku *tmp* o čom sa môžeme presvedčiť napríklad pomocou príkazu *ls*.

3. Celý virtuálny stroj už máme vytvorený, avšak stále v ňom nemáme knižnice, ktoré sú potrebné pre spustenie *Google unit frameworku*. Preto je potrebné priložiť

k tomuto virtuálnemu stroju aj potrebné knižnice. Ako bolo spomenuté v kapitole *Linkovanie headerov* o inštalácii *Google unit frameworku*, tak je potrebné všetky tieto knižnice nakopírovať do priečinku */tmp/lib*. Všetky zdrojové knižnice sa nachádzajú v priečinkoch */usr/lib* a */usr/include* alebo v ich pridružených podpriečinkoch.

4. Vzhľadom na to, že my potrebujeme pri spustení virtuálneho stroja spustiť skompilovaný súbor, tak mu prikážeme, aby pri svojom spustení ho vykonal, potom sa vypol a celý vymazal. Toto všetko docielime (okrem vymazania) pomocou zavádzacieho skriptu, ktorý sa bude nachádzať v priečinku */sbin/init*. V prípade ak sa tam nenachádza, tak ho vytvoríme napríklad pomocou príkazu *touch /sbin/init*. Do tohoto súboru vložíme nasledujúci text

```
1 #!/bin/sh
2 ./tmp/main.out --gtest_output=xml:./tmp/s.xml
3 poweroff
```

Táto sada príkazov nám zabezpečí spustenie testovaného súboru, zapísanie jeho výsledku do súboru v *XML* formáte a následné vypnutie virtuálneho stroju. Bližšie informácie o prepínačoch *Google unit frameworku* sa dajú nájsť v jeho dokumentácii[Goo].

4.4.2 Spustenie

Virtuálne prostredie budeme spúšťať vždy pri požiadavke na otestovanie zadania od užívateľov aplikácie. V predchádzajúcich kapitolách sme si uviedli zopár techník ako vytvárame a ukladáme súbory. Avšak toto nám nestačí, pretože je potrebné ich skompilovať a spustiť.

4.4.2.1 Kompilácia

Kompilácia zdrojových súborov prebieha pomocou kompilátora *g++*. Ide o kompilátor priamo zabudovaný vo väčšine linuxových distribúcií. My si tento kompilátor budeme spúšťať priamo z PHP skriptov a to pomocou príkazu

```
1 $error = shell_exec('g++ -I/home/gtest-1.7.0/include
2 -L/gtest-1.7.0/ /gtest-1.7.0/src/gtest_main.cc ' .
3 $includefiles . ' ' . $testfile . ' -lgtest -lpthread
```

```
-std=c++11 -o ' . $path .
'/test/tmp/main.out 2>&1 1>/dev/null');
```

Pomocou tohoto príkazu sa nám vytvorí nový skompilovaný súbor *main.out* v priečinku */main* v adresárovej štruktúre virtuálneho stroja (ako je spomenuté vyššie, tak práve tento súbor sa bude automaticky spúšťať pri vytvorení virtuálneho stroja). Avšak počas kompilácie môžu nastať chyby (napríklad syntaktické, nesprávne výsledné hodnoty funkcií a podobne), preto je potrebné odchytiť tento zlý výstup a to sa deje pomocou príkazu *2>&1 1>/dev/null*. Tento príkaz zabezpečí to, že v prípade chyby sa nám výstup dostane do výsledku funkcie *shell_exec* a vieme si ju odchytiť v PHP.

Ak kompilácia prebehne úspešne, tak nám zostáva vykonať predposledný krok testovania riešenia a to je spustenie riešenia. Riešenie sa podobne ako kompilácia spúšťa pomocou PHP skriptu

```
shell_exec('timeout 7s linux rootfstype=hostfs uml_dir=' .
$path . ' rootflags=' . $path . '/test rw mem=48M');
```

V tomto príkaze sa nachádzajú konfiguračné prepínače pre virtuálny stroj

- **timeout** pomocou tohoto príkazu ľahko nastavíme pridelený čas virtuálnu stroju, počas ktorého môže byť spustený
- **uml_dir** špecifikuje adresár, do ktorého si virtuálny stroj ukladá logy a ďalšie dáta potrebné pre jeho chod. Je potrebné ho nakonfigurovať vzhľadom na to, že príkaz je spúšťaný z PHP a samotné PHP, ktoré beží pod procesom *Apache Deamon*, nemá právo zapisovať do hociktorého priečinku v operačnom systéme, preto nastavíme na priečninok s riešením
- **rootfstype** príkaz deklarue, že virtuálny stroj bude používať ako súborový systém niektorý z podadresárov operačného systému
- **rootflags** predstavujú pre súborý systém virtuálneho stroja koreň
- **rw** deklaruje, že súborový systém budeme používať aj na čítanie aj zápis
- **mem** podobne ako príkaz *timeout* aj tento príkaz nám dovoľuje nastaviť prostredky pre virtuálny systém. Prostriedok, ktorý nastavujeme týmto príkazom je veľkosť pamäte, ktorú vyhradíme virtuálnemu stroju

4.4.2.2 Získanie výsledku z testovania

Posledný krok pri testovaní je získanie výsledku z testovania. Ako je spomenuté vyššie, tak výsledok z testovania sa ukladá do súboru vo formáte *XML*. Výsledok z neho získame pomocou nasledujúcej skupiny príkazov.

Najskôr si vložíme testovací súbor do komponentu *Nathanmac/Parser*. Tento komponent nám zabezpečí jednoduché spracovanie *XML* súboru.

```
1 $result = File::get($path . '/test/tmp/s.xml');
2 $parsed = Parser::xml($result);
```

V ďalšom kroku prebehneme cez všetky testovacie bloky, získame názov a prípadné chyby v bloku. Keďže jedna z požiadaviek na systém je, že má zobraziť len prvý zlyhaný test z každého bloku, tak užívateľ dostane informácie len o tomto prvom zlyhanom teste. Ak je blok bez chyby, tak si vytiahneme z databázy počet bodov za test a prirátame tieto body skupine.

```
1  foreach ($parsed['testsuite'] as $suite) {
2    if (isset($suite['testcase']))
3      $result .= 'BLOK: ' . $suite['testcase']['@attributes']['name'] .
           PHP_EOL;
4          if (isset($suite['testcase']['failure'])) {
5            $result .= '<pre style="color: red">' . $suite['testcase']['
                 failure'] . '</pre>' . PHP_EOL;
6          } else {
7            $task = Task::find($input['task_id']);
8            $block = $task->blocks()->get(['id'])->toArray();
9            $section = Section::whereIn('block_id', $block)->where('name
                 ', '=', $suite['testcase']['@attributes']['name'])->
                 first();
10           $points+= $section->points;
11         }
```

V prípade zlyhaného testu sa výsledok zobrazí užívateľovi ako text, kde je uvedená volaná funkcia aj s parametrami, jej výsledok a očakávaná hodnota. Ak pre zvolený blok všetky testy úspešne prešli, tak užívateľ dostane informáciu o bodovom ohodnotení bloku.

Záver

V tejto bakalárskej práci sme vytvorili webové prostredie pre online programovanie v skupine s možnosťou bezpečného testovania zdrojového kódu a jeho hodnotenie. Za výsledok tejto práce možno považovať funkčnú aplikáciu spĺňajúcu požiadavky, ktoré sme si stanovili. Aplikácia umožňuje vytváranie, editáciu zadaní, vytváranie testov ku zadaniu, vytvorenie predefinovaných súborov ku zadaniu, spúšťanie testovacích prípadov na serveri a samozrejme online spoluprácu na projekte, ktorú sme rozšírili do viacerých častí aplikácie ako je online chat v skupine alebo real-timová tvorba skupín k zadaniu.

Aplikáciu sme dokonca odskúšali v reálnej prevádzke počas hodiny Extrémneho programovania na Fakulte matematiky, fyziky a informatiky Univerzity Komenského. Od študentov sme dostali cenné rady a námety na vylepšenie určitých častí aplikácie, ale okrem toho poukázali aj na zopár menších chýb, ktoré sa v aplikácii v tej dobe nachádzali, čo bolo veľmi cenné k ďalšiemu vývoju.

Podľa odozvy študentov sa ukázalo, že aplikácia má určitý potenciál a môže byť reálne nasadená vo výučbe programovania a dokonca v komerčnom sektore s malými úpravami.

Aplikáciu je možné v budúcnosti rozširovať o rôzne komponenty, napríklad pridanie podpory pre ďalšie programovacie jazyky, ale nachádza sa tu veľký potenciál na využitie synchronizácie aj v iných komponentoch v aplikácii.

Literatúra

[Bec12] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley, 1st edition, 2012.

[Goo] Google. Dokumentácia google c++ testing frameworku. `https://code.google.com/p/googletest/wiki/V1_7_AdvancedGuide`. Navštívené 21.12.2014.

[Jur13] Andrej Jursa. Nový dlhodobý viacúčelový sklad zadaní. Bakalárska práca, Univerzita Komenského v Bratislave Fakulta matematiky, fyziky a informatiky, 2013.

[Les14] Pavol Lesčinský. Interaktívna webová učebnica programovania v c++. Bakalárska práca, Univerzita Komenského v Bratislave Fakulta matematiky, fyziky a informatiky, 2014.

[Sau14] Raphaël Saunier. *Getting Started with Laravel 4*. Packt Publishing, 1st edition, 2014.

[SH09] Nancy Holzner Steven Holzner. *Google Docs 4 Everyone*. Greg Wiegand, 1st edition, 2009.