SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

TÍMOVÝ PROJEKT

Monitorovanie programátora v IDE

Autori:
Bc. Michal JURANYI
Bc. Ivan KOŠDY
Bc. Jozef MARCIN
Bc. Tomáš MARTINKOVIČ
Bc. Matej NOGA
Bc. Ján PODMAJERSKÝ
Bc. Juraj RABČAN

Školiteľ:
Doc. Mgr. Daniela CHUDÁ, PhD.

2013/2014

POĎAKOVANIE

Celý tím ďakuje Doc. Mgr. Daniele Chudej, PhD. za jej vedenie a cenné rady počas práce na projekte.

Obsah

1 Úvod **1**

1.1 Celkový pohľad na projekt 1

2 Šprint 1 **3**

2.1 Analýzy vykonané v 1. šprinte 3

2.1.1 Niektoré súvisiace diplomové a bakalárske práce vedené na FIIT STU 3

2.1.2 Keystroke Dynamics 4

2.1.3 Perconik a UACA 6

2.2 Redmine 6

2.3 Webová stránka 7

3 Šprint 2 **8**

3.1 Analýza logerov 8

3.1.1 Eclipse Metrics plugin 1.3.8 8

3.1.2 Rabbit 9

3.1.3 Fluorite 10

3.2 Porovnávanie vektorov 10

3.3 Inštalácia Gitu 10

3.4 Inštalácia databázy na server 11

3.5 Pristúpenie k dátam z externej databázy 11

3.6 Spring security a registrácia s prihlásením sa do aplikácie 11

3.7 Inštalácia GlassFishu na server a nasadenie aplikácie 11

4 Šprint 3 **12**

4.1 Identifikácia jednotlivých prvkov vektora 12

4.1.1 diGraphs (character, flightTime, latency) 12

4.1.2 graphs 12

4.1.3 leftToRightClickPart 12

4.1.4 numClicks 12

4.1.5 numLeftClicks 12

4.1.6 numMaxAppl 13

4.1.7 numMiddleClicks 13
4.1.8 numMinAppl 13
4.1.9 numMoves 13
4.1.10 numRightClicks 13
4.1.11 numScrolls 13
4.1.12 numNormAppl 13
4.1.13 trigraphs 13
4.2 Vytvorenie agregovaného vektora 13
4.3 Implementácia grafu na zobrazovanie výsledkov 14
4.4 Implementácia filtra a zobrazenie vyfiltrovaných dát 14
4.5 Preverenie získavania dát 15
4.6 Globálny pohľad na program 15
4.7 Súčastná architektúra systému 16
4.8 Ganttov graf 17
4.9 Ganttov graf 18

Zoznam obrázkov

1 5
2 Ukážka ako pracuje Perconik 6
3 Ukážka webovej stránky 7
4 Ukážka grafu vo webovej aplikácii 14
5 Ukážka architektúry systému 16

1 Úvod

Dokument predstavuje technickú dokumentáciu k projektu Monitorovanie programátora v IDE. Projekt prebieha na Slovenskej technickej univerzite v Bratislave na Fakulte informatiky a informačných technológii v Bratislave. V tomto dokumente čítateľ nájde celkový pohľad na projekt, dokumentáciu k prebehnutým šprintom a aj ciele projektu.

1.1 Celkový pohľad na projekt

Naším cieľom je vytvorenie webovej aplikácie, ktorá umožní sledovanie aktivity programátora pracujúceho vo vývojovom prostredí. Naša aplikácia bude programátorov monitorovať v dvoch vývojových prostrediach a to v Eclipse a vo Visual Studiu. Hlavným dôvodom výberu týchto dvoch prostredí bol plugin Perconik, ktorý dokáže logovať aktivitu programátora v týchto dvoch prostrediach. Na základe nalogovaných dát budeme schopný identifikovať programátora, ďalej sa pokúsime určiť ako je efektívny. Plugin Perconik loguje dáta prostredníctvom UACA. UACA spája Perconik s databázou firmy Gratex International a. s. . Prostredníctvom webovej služby opísanej vo wsdl súbore pristupujeme k dátam z tejto databázy. Na základe týchto dát sme vytvorili model používateľa. Modely používateľa predstavujú vektory, ktoré sú ukladané do NoSql databázy MongoDB. Na implementáciu nášho riešenie sme zvolili jazyk Java 1.7 a framework Spring MVC. Toto rozhodnutie padlo aj preto, že každý člen tímu už s Javou robil. Ďalší dôvod bol ten ,že sa môžeme sami logovať a zbierať nové dáta.

Základná koncepcia riešenia:

- Logovanie dát
- Uspešné stiahnutie dát z Gratex databázy.
- Vytvorenie projektu v Spring MVC
- Nastavenie Spring security
- Nastavenie aplikačného servera

- Inštalácia MongoDB na server
- Implementácia vlastnej LOC metriky
- Vytvorenie modela používateľa a agregovaného vektora
- Vytvorenie filtra pre zobrazenie uložených modelov
- Vizualizácia údajov,ktoré sú nalogované pre konkrétneho používateľa

2 Šprint 1

2.1 Analýzy vykonané v 1. šprinte

2.1.1 Niektoré súvisiace diplomové a bakalárske práce vedené na FIIT STU

V tejto časti sa nachádza stručný prehľad niektorých diplomových a bakalársky prác vedených na FIIT STU, ktoré sa venujú problematike identifikácie používateľa. Témou identifikácie používateľa sa čiastočne venujeme aj v našom tímovom projekte Monitor programátora v IDE.

Model používateľa charakterizovaný dynamikou písania na klávesnici

V tejto diplomovej práci sa autor Rastislav Kršák zameriava na identifikáciu používateľa. Túto identifikáciu používateľa vyhodnocuje počas celej práce s počítačom. Priebežne sa vyhodnocuje, či v danom okamihu sedí za počítačom prihlásená osoba. Práca poskytuje prehľad o realizovaných výskumoch v predmetnej oblasti a ich výsledkoch. Popisuje výskum v oblasti dynamiky písania na klávesnici a práce s myšou. Vysvetľuje základné pojmy a metódy súvisiace s touto problematikou. Modelovanie používateľa vykonáva pomocou klávesnice a polohovacieho zariadenia. Výsledkom práce je implementovaný systém, ktorý dokáže priebežne sledovať činnosť používateľa s počítačom a na základe toho overiť alebo určiť jeho identitu. To zahŕňa vytvorenie modelu používateľa založeného na vybraných charakteristikách, ktoré boli získané extrakciou z údajov získaných pri práci používateľa s klávesnicou a polohovacím zariadením.

Model používateľa pre jeho identifikáciu

V tejto diplomovej práci sa autor Robert Godány venuje tvorbe modelu používateľa pre jeho identifikáciu na základe práca s myšou a klávesnicou. Výsledkom je vytvorenie modulu do webových aplikácií, ktorý bol integrovaný do Moodle. Identifikácia používateľa prebiehala pri písaní textu. V práci sa zameriaval na statické a dynamické charakteristiky.Extrahovali časy všetkých n-grafov, čo predstavuje čas od stlačenia prvej klávesy po stlačenie n-tej klávesy na klávesnici.

Podobnosti v slovenských textoch a v programových kódoch

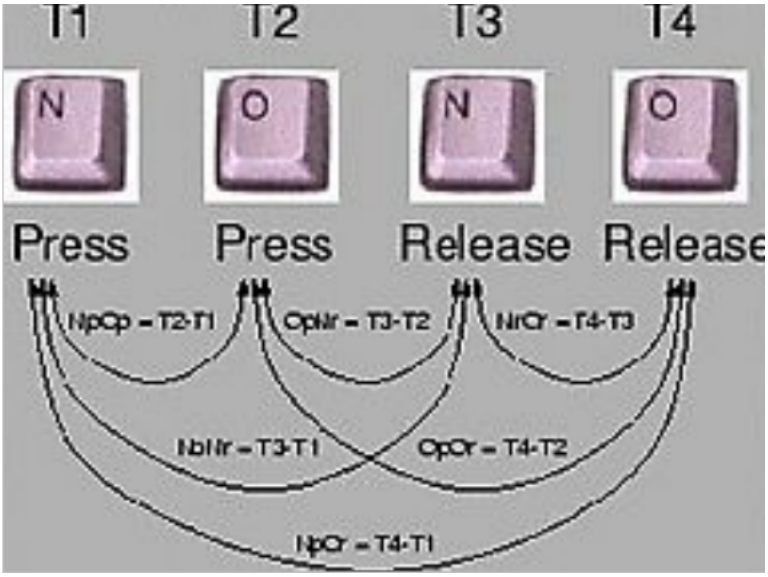
V tejto diplomovej práci sa autor Marián Hraško zameriava súčasne na plagiátorstvo v slovenských textoch a v zdrojových kódoch. Výsledkom sú navrhnuté transformácie programových štruktúr v jazyku C++ na jednotné tvary pre jednoduchšie nájdenie podobnosti v zdrojových kódoch.

Vplyv biometrických charakteristík na model používateľa pre identifikáciu

V tejto bakalárskej práci sa autor Peter Krajník zameriava na analýzu niekoľkých prác venujúcich sa biometrickým systémom. Tieto systémy sú zamerané na dynamiku klávesových úderov a pohybov myši.

2.1.2 Keystroke Dynamics

- Proces analyzovania spôsobu akým používateľ píše, pričom sa jeho aktivita sleduje v milisekundách. Ide o zaužívané rytmické patterny, ktoré svojím písaním vytvára.
- Veľa prác sa venovalo tejto metóde, ktorá sa prijala ako nosná a výskumníci ju považujú za jednu z najlepších spôsobov určenia identity.
- Každý používateľ si vytvára špecifický podpis svojím písaním.
- Nespomína vytvorenie používateľského modelu, ale modeluje ho na základe jeho rytmiky písania.
- Sleduje sa:
  - čas oneskorenia – čas medzi stlačením klávesy dole a jej pustením
  - čas letu – čas medzi stlačením párov kláves. Napríklad čas od stlačenia dvoch kláves po uvoľnenie ďalších dvoch kláves
  - čas stlačenia špecifických párov, trojíc, štvoríc(napr. e-a, a-b-c, d-a-n-o)
  - rôzne špecifiká
    - Či používateľ používa ľavý alebo pravý shift na napísanie L, alebo dokonca capslock



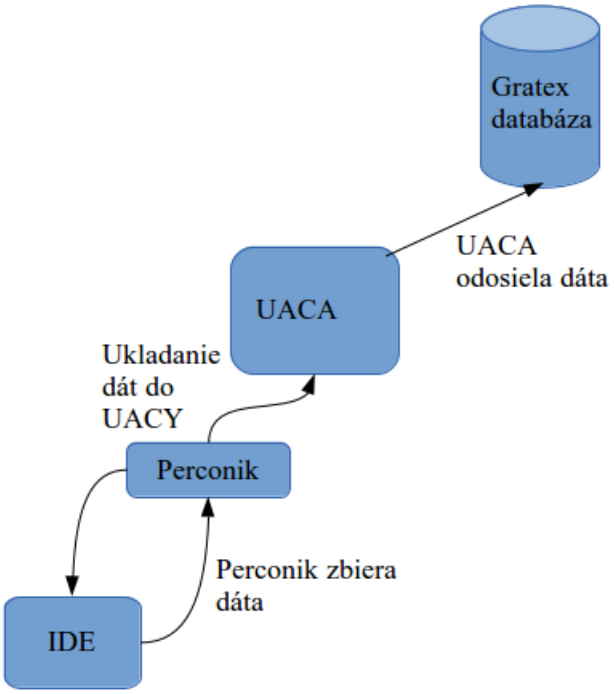
Obr. 1

∗ Štatistický výskyt chýb(kláves delete/backspace)

- Spôsoby vyhodnocovania :
- T – Test – Porovnáva sa štatistika dvoch sedení, nie veľmi presná(55%)
- Euclidova vzdialenosť – založený na porovnaní vzdialenosti medzi vektormi
  - $R = [r_1, r_2, ..., r_n]$ - prvý vektor
  - $R = [u_1, u_2, ..., u_n]$ - druhý vektor
  - $D(R, U) = [\sum_{i=1}^{N}(r_i - u_i)^2]^{\frac{1}{2}}$ - ak je nulová, zhoda je úplná
- Manhattanská vzdialenosť
- Cosínusová podobnosť
- Vážená vzdialenosť
  - Niektoré prvky sú viacej hodnoverné ako ostatné pretože pozostávajú z dlhších vstupov(nejakej vety napríklad).

2.1.3 Perconik a UACA

Perconik je plugin, ktorý používame na monitorovanie programátora. Plugin o programátorovi zachytáva rôzne informácie ako klikanie myšou či text, ktorý programátor skopíroval do IDE. Logy ktoré Perconik zaznamená ukladá do UACY. UACA slúži ako rozhranie medzi Perconikom a databázou, ktorá patrí Gratexu. UACA po uplinutí zvoleného časového intervalu odošle dáta databáze Gratexu. Pokiaľ chceme tieto dáta stianúť, môžeme tak urobiť za pomoci webovej služby opísanej vo wsdl súbore.



Obr. 2: *Ukážka ako pracuje Perconik*

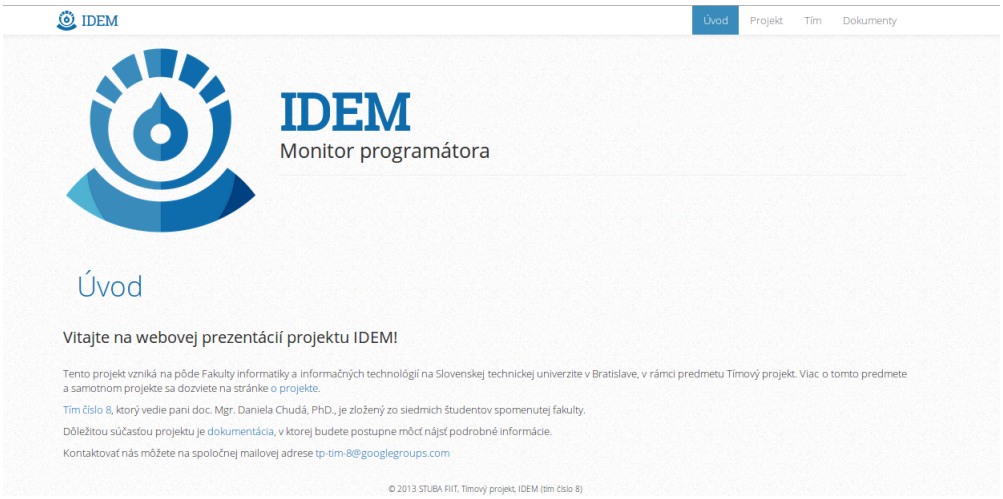
2.2 Redmine

Monitorovanie práce a manažment úloh je pri tímovej práci veľmi dôležitý a preto sme museli začať používať softvér, ktorý by náam to uľahčil. Z veľkého množstva sovtvéru, ktorý bol pre tento účel vytvorený, sme sa rozhodli použiť Redmine. Redmine nám beží na školskom serveri a môžeme ho navštíviť na adrese:

http://team08-13.ucebne.fiit.stuba.sk:443/

2.3 Webová stránka

Pre potrebu publikovania informácií o tíme bola vytvorená webová stránka. Stránka sa dá navštíviť na adrese: *http://labss2.fiit.stuba.sk/TeamProject/2013/team08is-si/index.html*. Na stránke sa dajú nájsť informácie o členoch tímu, dokumenty a informácie o projekte. Stránka je vytvorená staticky a obsahuje len HTML kód a kaskádové štýly.



Obr. 3: *Ukážka webovej stránky*

3 Šprint 2

3.1 Analýza logerov

3.1.1 Eclipse Metrics plugin 1.3.8

- Podpora pre Eclipse 3.5 a vyššie
- Sledujú sa programátorske metriky
  - *McCabe Cyclomatic Complexity* Sleduje počet riadiacich premenných (else, if, switch, while, ternarne operatory atď.). Robí priemer pre všetky metódy nachádzajúce sa v projekte. Vracia konkrétne číslo pre jednu metódu a to je maximálne číslo z existujúcich metód.
  - *Number of Parameters* Počet parametrov metód v jednotlivých metódach, priemerný počet, maximálny počet.
  - *Nested Block Depth* Koľko krát sa v metóde volá konkrétna iná metóda. Tak isto obsahuje aj priemerný počet a maximálny počet
  - *Afferent Coupling(CA)* Počet tried mimo balíčka, ktoré závisia na triedach vnútri balíčka
  - *Efferent Coupling(CE)* Počet tried v balíčku, ktoré závisia na triedach v inom balíčku
  - *Abstractness(RMA)* Počet abstraktných tried (a interfejsov) delený celkovým počtom typov v balíčku
  - *Normalized Distance*. Predstavuje hodnotenie dizajnu balíčkovania. Čím je číslo menšie, tým lepšie. Jej hodnotu vypočítame $|RMA + RMI - 1|$
  - *Total Lines of Code* Celkový počet neprázdnych riadkov, nezaratáva komentáre.
  - *Number of Interfaces* Počet interfejsov.
  - *Number of classes* Počet tried.
  - *Number of Static Methods* Počet statických metód.

  - *Number of Static Methods* Počet metód v triede.

- Tento plugin nesleduje
  - počet znakov za daný čas
  - počet klikov myši
  - celkový čas strávený tvorbou programu
  - metriky vývojového prostredia(kto kopíroval odkiaľ kam, aké príkazy používateľ používa najčastejšie atď.)
- Metriky sa nezaznamenávajú periodicky, ale ak sa niečo uloží, alebo ak sa prepne myšou medzi triedami, tak až potom sa zaznamenávajú. Metriky sa dajú exportovať do XML súboru, avšak nie je to automatizované.
- Ak chceme XML vytvoriť, stačí kliknúť na tlačidlo XML aplugin XML vytvorí a vyexportuje, kde používateľ chce. Avšak, metriky je možné pozerať si v samostatnom view, ktorý plugin do Eclipse vytvorí. V rámci tohto view si používateľ môže jednotlivé hodnoty prezerať v reálnom čase.
- Plugin nie je možné voľne upravovať.

3.1.2 Rabbit

Plugin je podporovaný v Eclipse 3.4 až 3.6. Tento plugin sa už aktívne nevyvíja. Zobrazenie údajov je možné len v eclipse, pretože sa neukladajú do XML súbora. Sledujú sa metriky časového použitia Eclipse a to:

- čas strávený kompilovaním
- čas strávený používaním jednotlivých perspektív
- čas práce na úlohách
- tvorbou nejakej metódy alebo triedy
- Používané príkazy(cut, copy, paste)

3.1.3 Fluorite

Plugin loguje character typing:

- Posúvanie kurzora
- Zmeny označeného textu
- Loguje príkazy a aj parametre príkazov
- Kopírovanie textu s uvedenými hodnotami odkiaľ, kam a dokonca aj čo
- Loguje zmeny dokumentu

Dáta sa logujú do XML, pričom tieto XML súbory sú strašne veľké. Za 5 minút skúšania sa vygeneroval XML súbor, ktorý mal viac ako 700 riadkov. Nepodarilo sa nájsť dokumentáciu o tom ako sa XML súbor má správne rozparsovať. Prístup k logom je možný až po ukončení Eclipse. Logujú sa aj skratky používané v Eclipse, napríklad Ctrl + Space. Logujú sa aj eventy ako otvorenie triedy v prehliadači súborov. Zaloguje to, ako presne trieda vyzerá, importy balíčkov, tried atď.

3.2 Porovnávanie vektorov

V tomto šprinte bola vytvorená trieda, ktorá obsahuje metódy na porovnávanie vektorov. Nech P a Q sú vektory, $p_1...p_n$ sú prvky vektora P a $q_1...q_2$ sú prvky vektora Q.

- *Euclidová vzdialenosť*: $d(Q,P) = \sqrt[2]{|q_1 - p_1| + |q_2 - p_2| + ... + |q_n - p_n|}$
- *Manhattanská vzdialenosť* $D(Q,P) = |q_1 - p_1|^2 + |q_2 - p_2|^2 + ... + |q_n - p_n|^2$
- *Manhattanská vzdialenosť*$(QP) = \frac{\sum_{i=1}^{n} \mathbf{q}_i \mathbf{p}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{q}_i)^2} \sqrt{\sum_{i=1}^{n} (p_i)^2}}$

3.3 Inštalácia Gitu

V tomto šprinte bolo načase vybrať si jeden z mnoha nástrojov, ktorý by nám umožnil správu zdrojového kódu. Rozhodnutie padlo na necentralizovaný a distribuovaný systém riadenia revízií GIT. GIT bol nainštalovaný na server a každý člen tímu si ho naištaloval lokálne. Pôvodne boli vytvorené dve vetvy: *Master* a *develop*. Neskôr si každý člen tímu vytvoril svoju vetvu.

3.4 Inštalácia databázy na server

V tomto šprinte sme sa rozhodovali aku databázu budeme používať. Po dlhej diskusii sme sa rozhodli pre MongoDB. MongoDB je NoSql databáza s dobrou podporou vo frameworku Spring, ktorý používame pri vývoji aplikácie. Databázu si každý musel naištalovať aj lokálne, pretože aktuálna verzia aplikácia vyžaduje databázu na stroji, kde aplikácia beží.

3.5 Pristúpenie k dátam z externej databázy

Prostredníctvom webovej služby vieme dostať údaje z databázy Gratexu. Problém je, že tieto údaje sú dosť neprehľadné a ťažko sa v nich orientuje. Preto je potrebná analýza týchto údajov.

3.6 Spring security a registrácia s prihlásenïim sa do aplikácie

Počas šprintu došlo k nastaveniu Security Springu, čiže používateľ, ktorý má obmedzené práva môže stránku vidieť inakšie ako používateľ s plnými právami. Bol vytvirený registračný formulár, ktorý umožní používateľovi registrovať sa do aplikácie a bol vytvorený aj druhý formulár, ktorý používateľovi umožní prihlásenie do aplikácie.

3.7 Inštalácia GlassFishu na server a nasadenie aplikácie

Na tímový server bol nainštalovaný GlassFish 4. Po jeho nainštalovaní aplikácia nasadená na tímový server.

4 Šprint 3

4.1 Identifikácia jednotlivých prvkov vektora

Po analýze sme identifikovali nasledujúce prvky vektora:

4.1.1 diGraphs (character, flightTime, latency)

1. character – Dvojica písmen napr. „AB“, „RC“, atď.
2. flightTime – čas, milisekundy
3. latency – čas, milisekundy

Keďže prakticky každý model má viacero dvojíc, niekedy spoločných, niekedy dokonca úplne rozdielnych, vybrať reprezentatívnu zložku nebude práve najlepšie. Preto vyberieme priemer hodnôt.

Položka vektora V1 sa teda vyráta takto: $V_1 = 1/n \sum_(i = 0)^n P_i$ Kde – n je počet nenulových položiek v liste diGraphs. A P_i je číslo vyrátané z hodnôt:

$P_i = (flightTime + latency)/2$

4.1.2 graphs

podobne ako 1., rovnaká trojica, avšak character sa skladá len z 1 písmena. Ráta sa teda rovnako ako 1. Položka vektora V2 - je teda double číslo vyrátane rovnako avšak $P_i = (flightTime + latency)/1$

4.1.3 leftToRightClickPart

V3 = double číslo skopírované z tejto položky.

4.1.4 numClicks

Počet klikov, V4 = skopírované číslo z tejto položky

4.1.5 numLeftClicks

Počet klikov ľavým tlačidlom, V5 = skopírované číslo z tejto položky

4.1.6 numMaxAppl

len skopírujeme toto číslo do nášho vektora, konkrétne do položky V6

4.1.7 numMiddleClicks

Počet klikov stredným tlačidlom – iba skopírujeme do položky V7

4.1.8 numMinAppl

Počet minimalizovaných aplikácií, iba toto číslo skopírujeme do položky V8

4.1.9 numMoves

Iba skopírujeme do položky vektora V9

4.1.10 numRightClicks

Iba skopírujeme do položky vektora V10

4.1.11 numScrolls

Skopírujeme do položky vektora V11

4.1.12 numNormAppl

Počet spustených aplikácií, skopírujeme do položky vektora V12

4.1.13 trigraphs

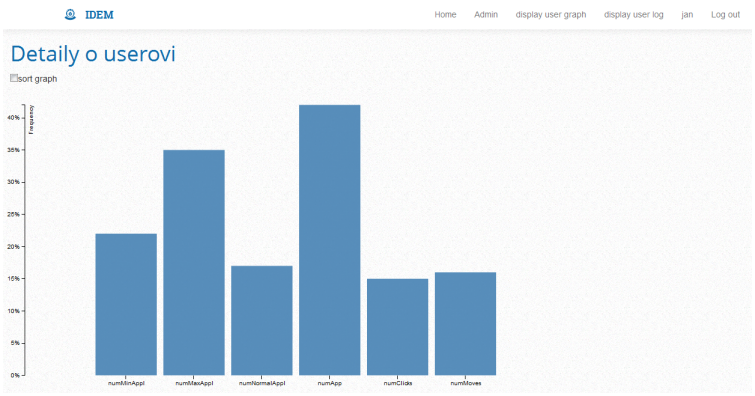
Podobne ako bod 1., avšak vzorce sú: $P_i = (flightTime + latency)/3\ V12 = 1/n \sum_(i = 0)^n P_i$

4.2 Vytvorenie agregovaného vektora

Po identidikovaní prvkov vektora, boli vytvorené triedy ktoré poskytovali tento vektor. Ako vstup pre tieto triedy slúži trieda model používateľa.

4.3 Implementácia grafu na zobrazovanie výsledkov

Vytvorili sme triedu na grafickú reprezentáciu vektora madelu používateľa. Táto trieda zobrazí všetky hodnoty vektora na grafe. Graf zobrazuje hodnoty dynamicky, pri zobrazení sa hodnoty najprv zobrazí a v štandartnom poradí, ale po pár sekundách sa tieto hodnoty utriedia od najväčšej po najmenšej, tak poskytnú kompletný prehľad.



Obr. 4: *Ukážka grafu vo webovej aplikácii*

Tento graf je naprogramovaný v javascripte v d3 frameworku. Hodnoty na zobrazenie sú poslané do metódy, ktorá ich jednoducho spracuje a zobrazí na grafe.

4.4 Implementácia filtra a zobrazenie vyfiltrovaných dát

Potom ako sme mali dostatok údajov mohli sme vizualizovať dáta. Preto bol implementovaný filter, ktorý umožní vybrať agregované vektory používateľov, pričom sa môže filtrovať podľa dátumu, maximálnej dĺžky logu, minimálnej dĺžky logu a ešte aj podľa používateľského mena v Perconiku. Podľa toho aký používateľ zadá filter sa vytvorí najskôr query do MongoDB databázy a vytiahne všetky agregované vektory pre používateľa, ktorý spĺňa kritéria filtra. Vyťahuje sa tak, aby bol najmladší vektor na prvom mieste listu, ktorý vráti databáza. Ďalší krok je výpočet podobnosti medzi vektormi. Potom ako sa tento proces vykoná, pošle sa do JSP stánky atribút pomocou anotácie, ktorá je na to určená. V JSP sa pomocou iterátora zobrazí obsah vrátených vektorov aj s vypočítanou podobnosťou do tabuľky.

4.5 Preverenie získavania dát

V predchádzajucom šprinte sme mali problém, lebo aj napriek tomu, že sme logovali dáta z IDE, tak nám ich volaná služba nikdy nevrátila. Vrátila len dáta mimo IDE. Po analýze a konzultácii s tvorcami Perconika a UACY sme zistili, že služba dáta vracia, ale my ku ním nepristupujeme. Po miernej úprave aplikácie sa už k dátam vieme dostať.

4.6 Globálny pohľad na program

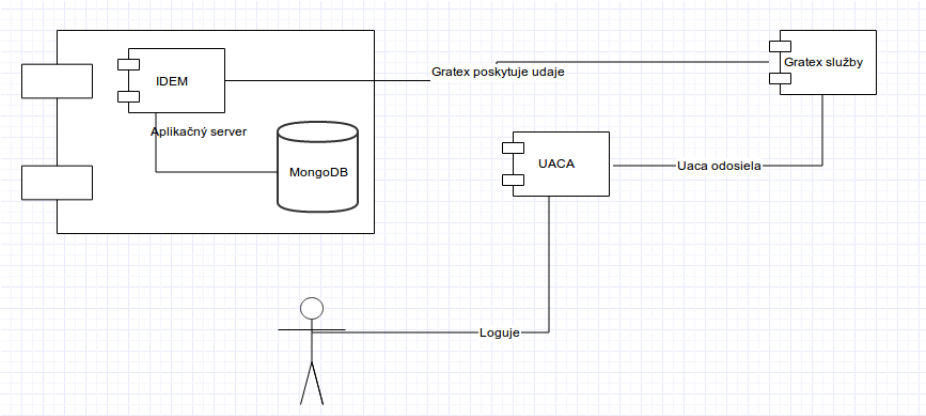
Aplikácia je vyvíjaná v Jave 1.7 a frameworku Spring MVC. Vzhľadom na technológie a webový charakter aplikácie sme sa ju rozhodli nasadiť na aplikačný server GlassFish. Tento server už bol nainštalovaný aj na náš tímový server. Aplikácia pracuje s NoSql databázou MongoDB.

- Naša aplikácia má momentálne implementované prepojenie s databázou z Gratexu. Vieme z nej prebrať údaje aj poslať ďalšie údaje, ktoré sa do nej zapíšu.
- Na logovanie programátora používame plugin Perconik.
- Do aplikácie boli implementované algoritmy, ktoré slúžia na porovnávanie vektorov a to euklidova vzdialenosť, manhattanská vzdialenosť a kosínusová vzdialenosť.
- Podľa údajov, ktoré dostávame z externej databázy sme identifikovali model používateľa. Podľa tohto modela bola vytvorená trieda, ktorej úlohou je vytvorenie agregovaného vektora z modela používateľa.
- Do aplikácie sa môže používateľ registrovať a prihlásiť. Bolo nastavené Spring security a autentifikácia.
- Sú vytvorené dve okná slúžiace najmä pre debugovanie. Jedno okno zobrazuje surové dáta z externej databázy a druhé zobrazuje modely používateľa podľa zadaného mena pričom dáta berie tiež z externej databázy.

- V aplikácii je okno v ktorom je filter a po jeho vyplnení sa zobrazia agregované vekory, ktoré prislúchajú konkrétnemu používateľovi. Pri vypĺňaní filtra sa môže vybrať aký druh podobnosti sa bude s vektorov počítať.

4.7 Súčastná architektúra systému

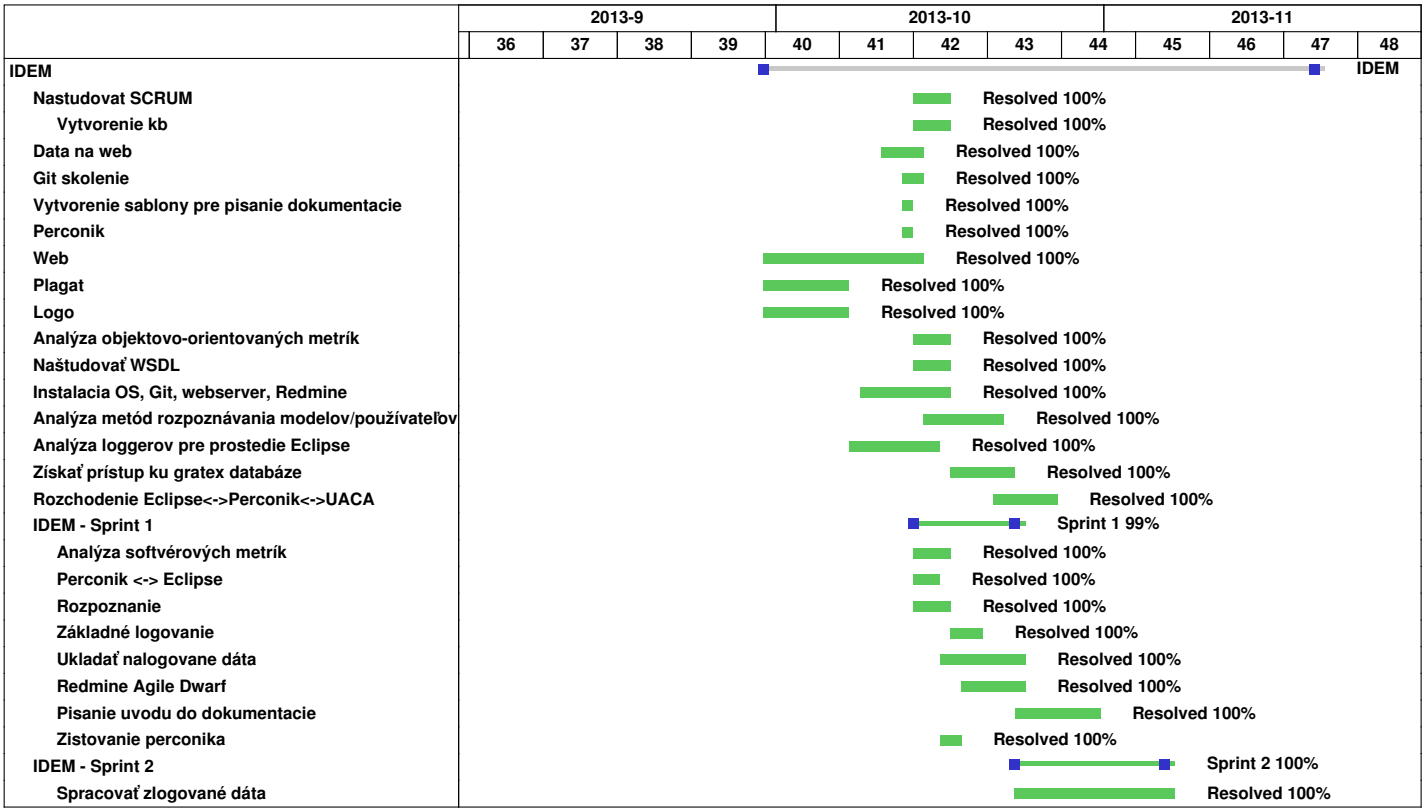
Súčastná architektúre systému je vyjadrená nasledujúcim obrázkom:



Obr. 5: *Ukážka architektúry systému*

4.8 Ganttov graf

IDEM



4.9 Ganttov graf