Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 19 Bratislava 4

Prepájanie dát o vývoji softvéru

Dokumentácia produktu

Členovia tímu č. 8:	Bc. Peter Bobovský
	Bc. Michal Kráľ
	Bc. Peter Kučera
	Bc. Marek Mura
	Bc. Miriama Pomffyová
	Bc. Lukáš Račko
	Bc. Michal Slovík

Predmet:	Tímový projekt II
Vedúci:	Ing. Martin Konôpka
Akademický rok:	2016/2017

1. Big picture

1.1 Úvod

V tejto časti sa venujeme projektu ako cieľovému produktu našej práce. Popisujeme tu čo je hlavným cieľom projektu, návrh riešenie a následnú implementáciu.

Programátori, rovnako ako aj my, pri tvorbe softvéru pracujú s veľkým množstvom podporných systémov, ktoré im pomáhajú pri vývoji. Medzi najznámejšie patria vývojové prostredia (Eclipse, Visual Studio), systémy pre správu zdrojových kódov (Git), úloh a nahlasovania chýb (Bugzilla, Jira, Redmine), prehliadok zdrojového kódu (Gerrit), alebo aj diskusné fóra a dokumentácie (StackOverflow, MSDN). Ako sme už opísali v motivácii, problém nastáva, keď sa snažíme nájsť súvislosti medzi nimi. Príkladom je prepojenie otázky v StackOverflow od vývojára pracujúceho na úlohe z Bugzilla a jeho commity v Git. Pri následnej prehliadke zmien v Gerrit má kontrolór pred sebou iba výsledný návrh zmeny v Git a nie odpoveď zo StackOverflow, na ktorej vývojár svoje riešenie založil.

Cieľom tohto projektu bude vytvoriť zberač dát z jednotlivých podporných systémov a následne ich spracovať tak, aby nadobúdali prínosnú hodnotu ako pre samotného programátora, tak aj pre manažéra projektu a ostatných členov tímu. Každý účastník vývoja bude mať dokonalý prehľad o tom, ku ktorému „Task“ patrí ktorý „Code Review“, „Activity“ a „Source Code“, poprípade, „Knowledge about Software“. Takýmto spôsobom sa naplní význam názvu projektu – TRACKS.

1.2 Globálne ciele projektu

Hlavným cieľom projektu je vytvorenie systému TRACKS pre zber a prepájanie dát, ktoré po sebe zanechali vývojári pri vývoji softvéru, t.j. „Tasks, Code Reviews, Activities, Source Code and Knowledge about Software“. Zber dát by mal byť vykonávaný pomocou botnet klientov na základe zadaní od hlavného uzla. Nasleduje prevedenie dát do spoločnej reprezentácie, ich samotné prepájanie a sprístupnenie tretím stranám. Jedným z hlavných cieľov je navrhnúť infraštruktúru tak, aby bola ľahko nasaditeľná ako na fakulte, tak aj interne v softvérovej firme a bola využitá záverečnými projektmi na fakulte.

1.2.1 Výsledky za zimný semester

V priebehu zimného semestra sme pracovali na splnení krokov, vďaka ktorým budeme vedieť dosiahnuť stanovené ciele tímového projektu.

TripleStore
Jedným z krokov, ktoré sme splnili bolo implementovanie základnej funkcionality modulu TripleStore. Spojazdnili sme RDF databázu, ktorá umožňuje ukladať zozbierané dáta z klientov. Dáta sa ukladajú v trojiciach, medzi ktorými je možné vykonávať sémantické vyhľadávanie.

TripleStore poskytuje API pre klientov, aby mohli nahrávať trojice zo zozbieraných dát. Klient

vytvorí inštanciu Graph a tá sa posiela na server. Server validuje prichádzajúce trojice a pokiaľ spĺňajú stanovené ontológie, trojice sa vložia do RDF databázy na serverovej strane.

Balancer
Je úspešne implementovaný v rámci servera. Má základnú funkcionalitu, ktorá podporuje komunikáciu s klientami a vytvára joby pre jednen typ identifikovanej stratégie.

SQL Databáza
Podarilo sa nás úspešne implementovať a nasadiť databázový server MSSQL. Schéma databázy zodpovedá doterajším požiadavkám aplikácie. Databáza je prístupná zo servera.

1.2.2 Ciele na letný semester

Na základe doterajšej práce na projekte, sme si stanovili ciele na letný semester.

TripleStore
V letnom semestri by sme chceli prepojiť existujúca funkcionalitu v TripleStore s klientom a dôkladne otestovať API na reálnych dátach, aby spoľahlivo ukladalo dáta v TripleStore databáze.

Balancer
Doplniť širšiu funkcionalitu v rámci komunikácie medzi serverom a klientom. Identifikovať stratégie pre iné repozitáre ako sú Stackoverflow a Github. Implementovať tieto stratégie a rozšíriť funkcionalitu aktuálnych.

SQL Databáza
V prípade, že sa budú meniť požiadavky na celkový produkt tak prispôsobiť schému DB.

Executables

Doplniť interpretáciu konfiguračných súborov ako aj zjednotiť komunikačný systém s klientom.

Dôležité termíny:

- február 2017 **-** Odovzdanie priebežnej správy o riešení projektu vo forme rozšíreného abstraktu na IIT.SRC.
- 27. apríl 2017 - Ukážka projektu v rámci študentskej vedeckej konferencie IIT.SRC 2017.

1.2.3 Výsledky za letný semester

TripleStore
V letnom semestri sa nám podarilo prepojiť funkcionalitu TripleStore s klientom, ktorý

úspešne stiahnuté dáta v RDF trojiciach posiela balanceru v podobe IGrafov a tie sú následne ukladané do TripleStoru. Dáta sa ukladané podľa jednotlivých projektov a ich repozitárov.

Balancer
Na základe analýzy bola vytvorená len jedna generická stratégia, ktorá je vhodná pre všetky doterajšie repozitáre.
Bola doplnená funkcionalita v rámci komunikácie medzi klientom a serverom, napr.inteligentné prideľovanie jobExecution.

SQL Databáza
Požiadavky boli rozšírené o ďalšie časti ako je napríklad Logovanie crash logov od klienta a databáza bola upravená podľa týchto požiadaviek.

Klient
Pomocou klienta sa nám podarilo stiahnuť úspešne prvé dáta. Taktiež bola vytvorená verzia inštalácie one-click, ktorá umožňuje jednoduchú inštaláciu klienta na ľubovoľný počítač. Taktiež náš klient prešiel testovaním.

Executables

Jednotlivé executables alebo DLL boli dotiahnuté do plnej funkcionality sťahovania. Bol vyrobený jednotný štandard na implementovanie DLL, ktorého súčasťou je základná "naivná" implementácia. V tejto implementácii je definovaný štandard komunikácie s klientom a serverom.

1.3 Celkový pohľad na systém

Kľúčovými komponentmi systému TRACKS pre vytvorenie sémantickej vrstvy nad dátami o vývoji softvéru by mali byť:

Balancer – hlavný (master) uzol pre plánovanie zberu dát, dáva Daemonovi informácie o tom, čo sa má v akých intervaloch sťahovať a posielať od klienta.

Klient API (Daemon) – distribuovaní botnet klienti pre zber dát. Po prijatí príkazu z Balancera spúšťa úlohu (job). Job bude obsahovať parametre pre konfiguračný súbor. Takto univerzálnemu klientovi nastavíme, čo chceme sťahovať.

DLL knižnice určené na sťahovanie - Pomocou týchto knižníc sa budú sťahovať jednotlivé repozitáre.

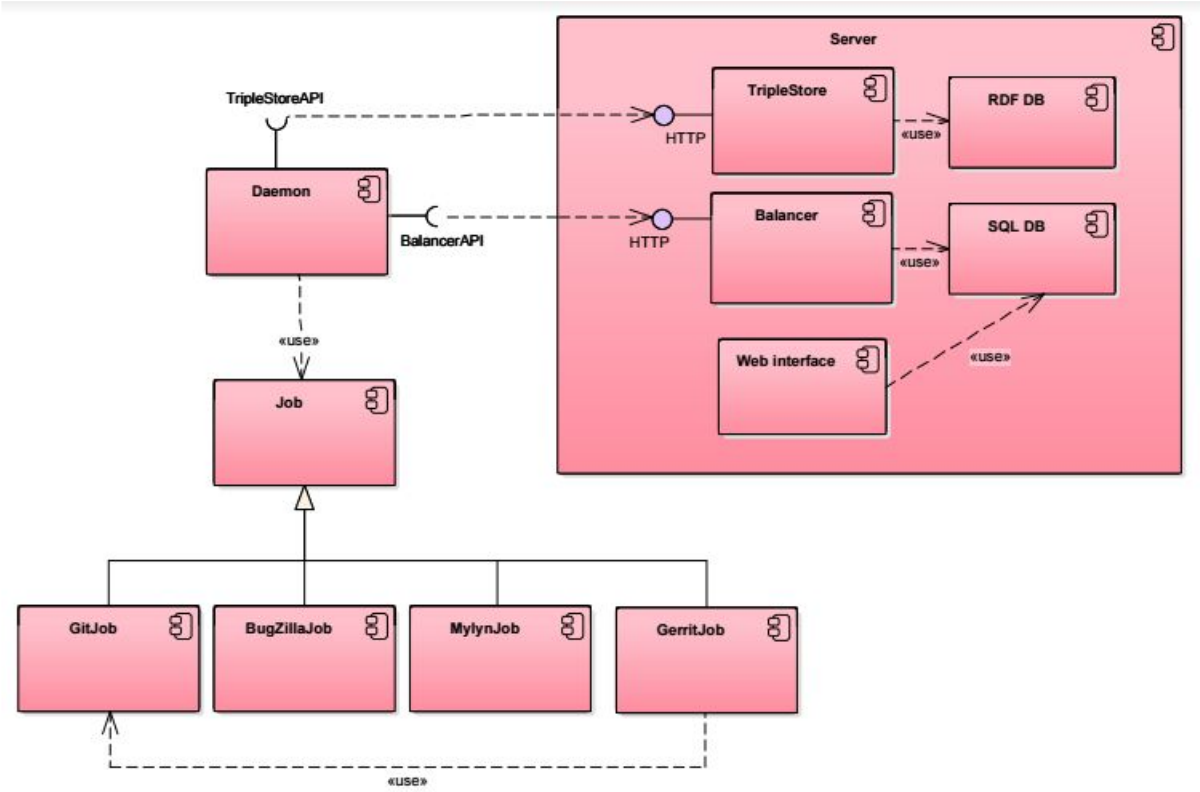
Webový server - poskytuje používateľovi-adminovi pridávať a upravovať jednotlivé projekty a dáta k nim.

MSSQL DB - databáza na uchovávanie dát o projektoch a klientoch.

Triplestore – umožní pripojenie na databázu so zozbieranými dátami z repozitárov a

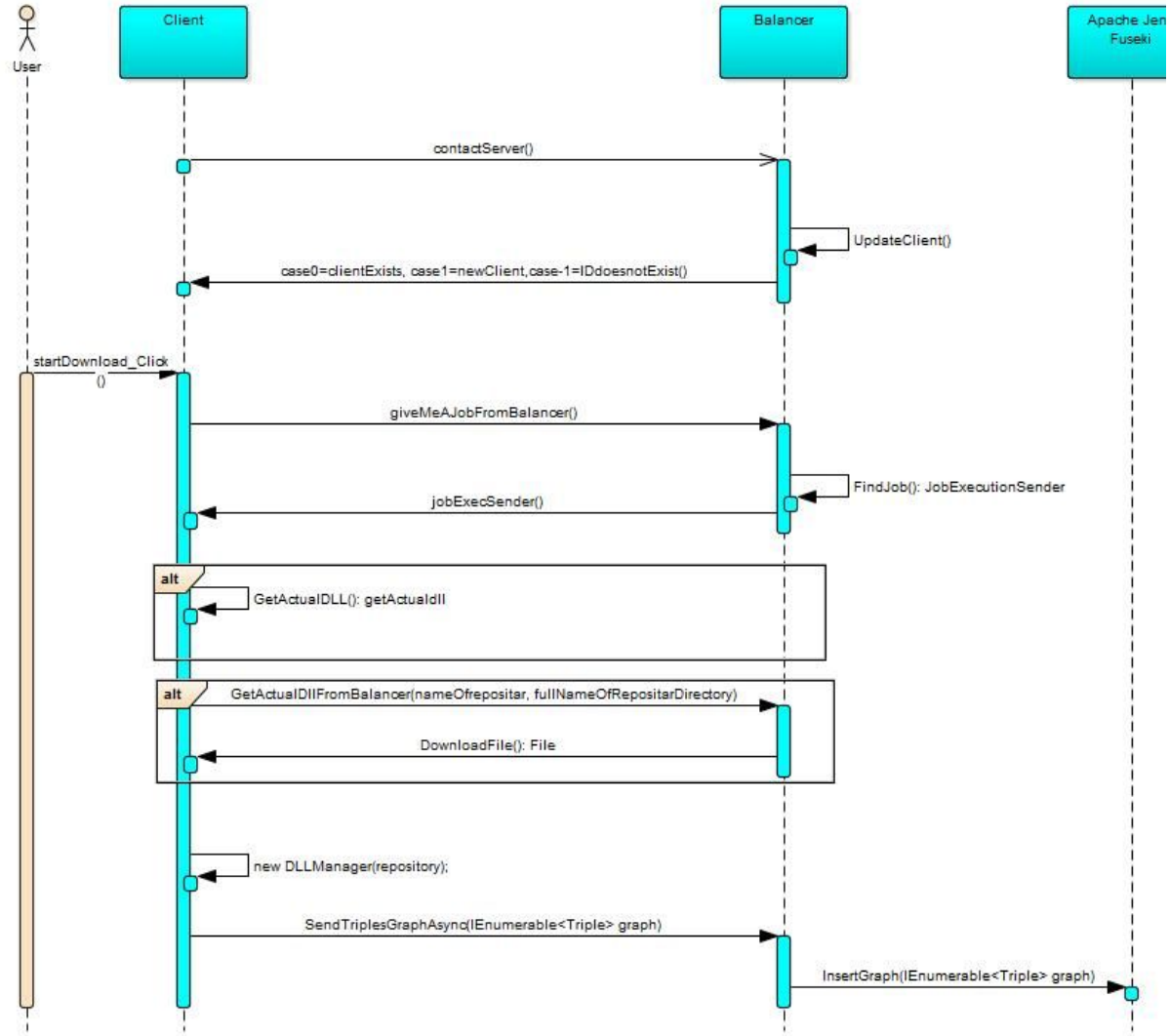
odvodenými metadátami. Ukladanie dát je v trojiciach – RDF (Resource Description Framework).

Nasledujúci diagram komponentov zobrazuje ich poprepájanie.



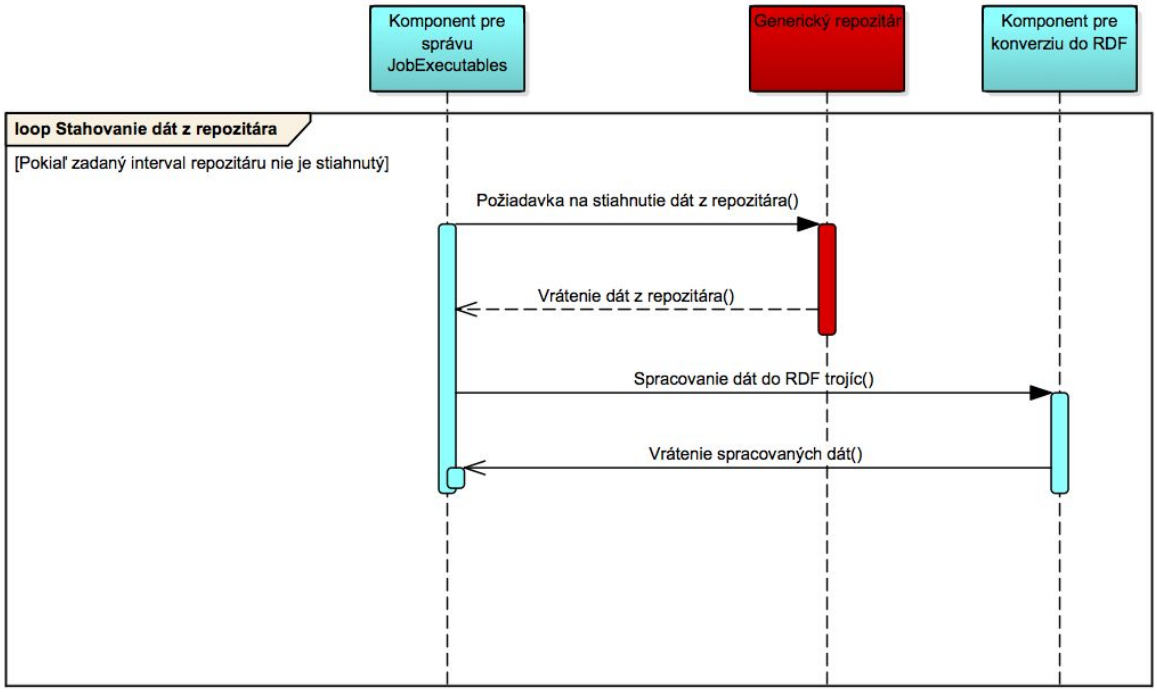
Obrázok 3 Diagram komponentov

Podstatná časť fungovania systému je zachytená na nasledovnom sekvenčnom diagrame. Znázorňuje priebeh prihlásenia klienta na server, pridelenia úloh, vykonania úloh a následne odoslanie spracovaných dát na server.



Obrazok č. 4 - Diagram sekvencií znázorňujúci fungovanie systému

Detail komponentu, ktorý zabezpečuje vykonanie JobExecutable, spracovanie dát a odoslanie klientovi sa nachádza na nasledovnom sekvenčnom diagrame.



Obrazok č.5 - Diagram sekvencií znázorňujúci sťahovanie dát z repozitára

1.4 Server

1.4.1 Triple Store

Analýza modulu TripleStore
Triplestore je samostatný modul na serverovej strane, ktorý umožňuje spracovávať prijaté data od klientov a uchováva ich v špecializovanej databáze. Implementovali sme knižnicu Apache Jena Fuseki, ktorá umožňuje uchovávanie dát v RDF. Dáta sú reprezentované v trojiciach – Subject – Predicate – Objekt.

- Subject reprezentuje identifikátor trojice
- Predicate hovorí, aký typ informácie obsahuje daná trojica
- Object obsahuje samotnú hodnotu trojice

Každá trojica reprezentuje konkrétnu hodnotu. V porovnaní s SQL databázou, Subject v RDF je podobný ako primary key v SQL, presne identifikuje danú trojicu. Predicate je ako názov stĺpca v SQL, určuje, aký typ hodnoty v sebe daná trojica uchováva. Object obsahuje konkrétnu hodnotu.

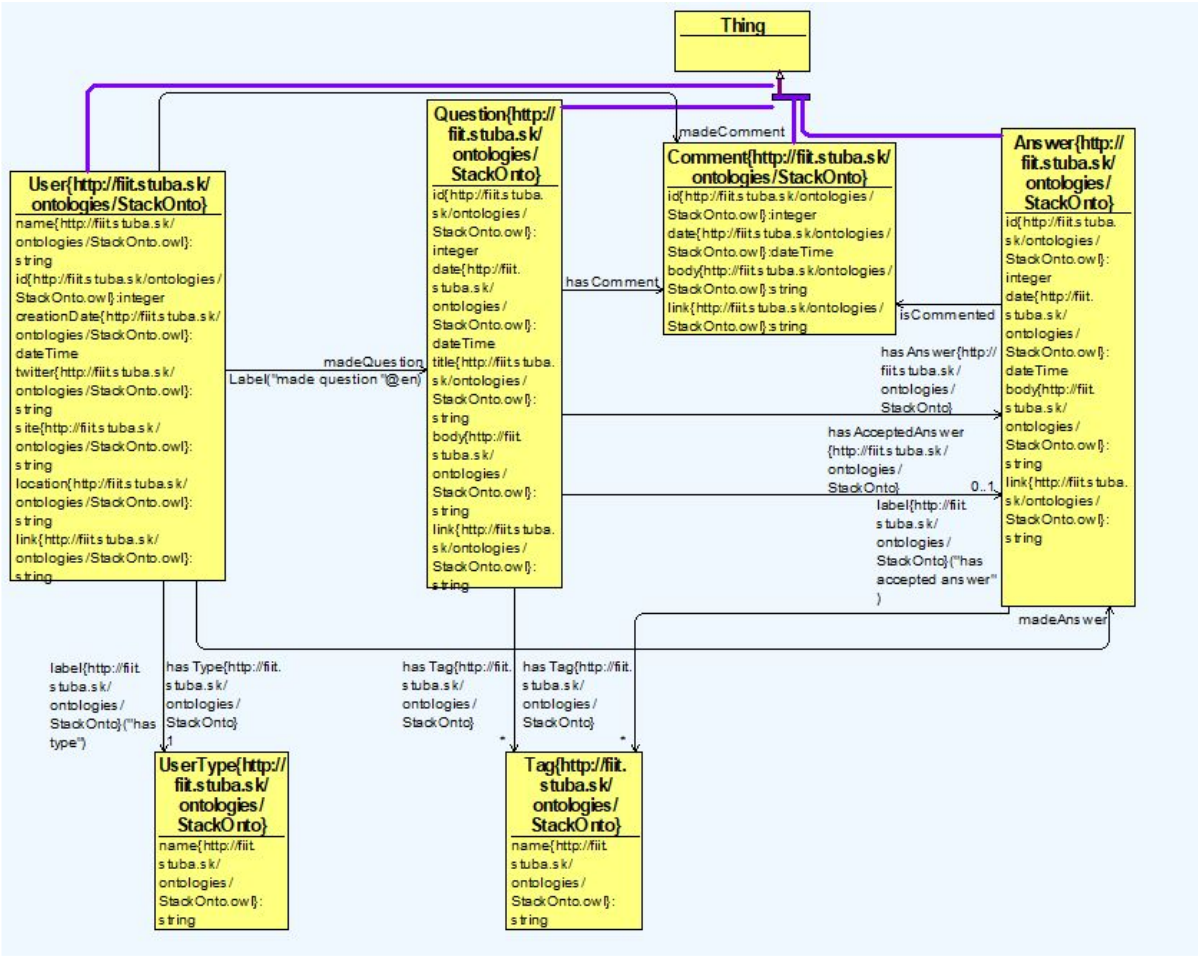
Dôvod, prečo je potrebné reprezentovať získané dáta v trojiciach súvisí s potrebou vykonávať sémantické vyhľadávanie nad zozbieranými dátami v RDF databáze. Vytvorené trojice sa pred poslaním na server vložia do tzv. IGrafu.

Návrh modulu TripleStore

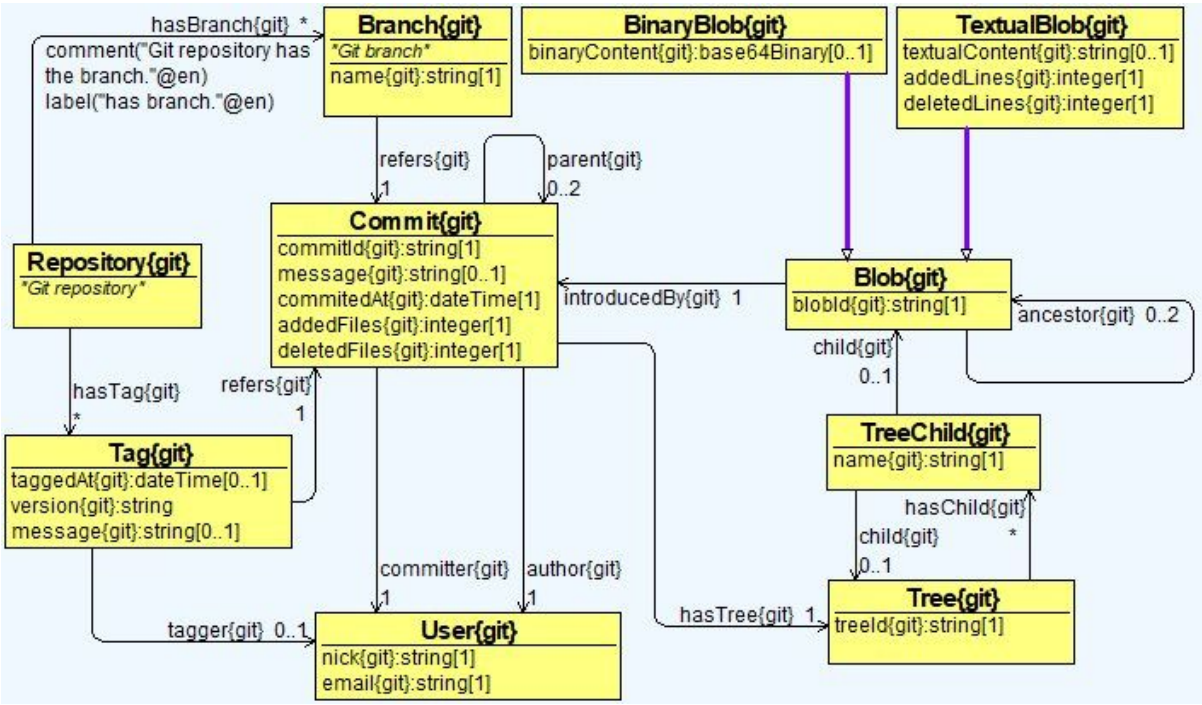
Modul TripleStore je súčasťou servera, ktorý umožňuje zadávať úlohy pre klientov a následne zbiera získavané dáta. Poskytuje API pre klientov, ktoré im umožňuje nahrávať vytvorené trojice do databázy Jena.

Klienti vytvárajú trojice na podľa vytvorených ontológií pre každý typ repozitáru. Tieto ontológie predpisujú, aké dáta majú trojice obsahovať. Pomocou rovnakých ontológií sa na serverovej strane v TripleStore validujú prichádzajúce dáta a odfiltrujú sa tie, ktoré nespĺňajú požadované ontológie. Výsledný graf s vyhovujúcimi trojicami sa následne uloží do databázy Jena.

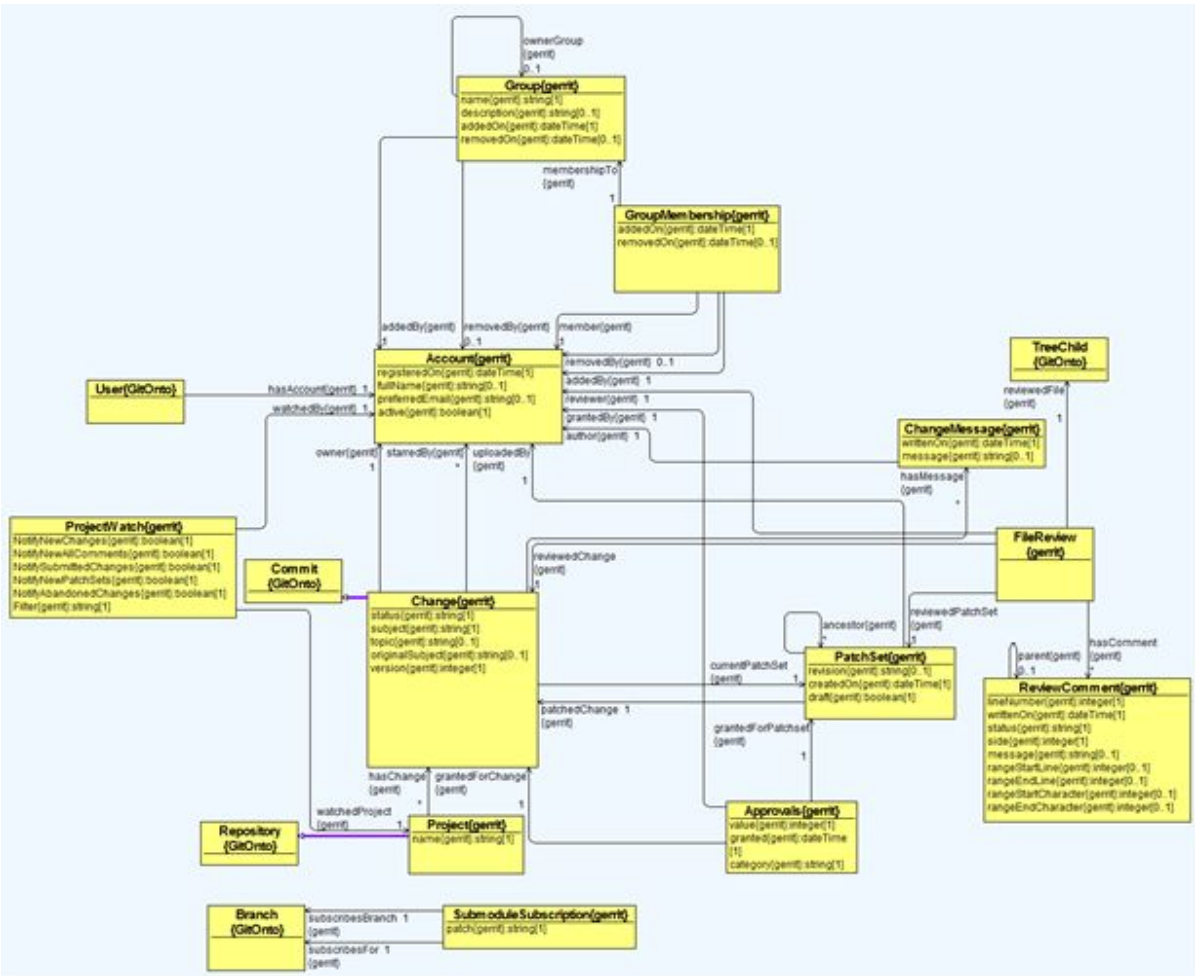
Ontológia pre Stackoverflow



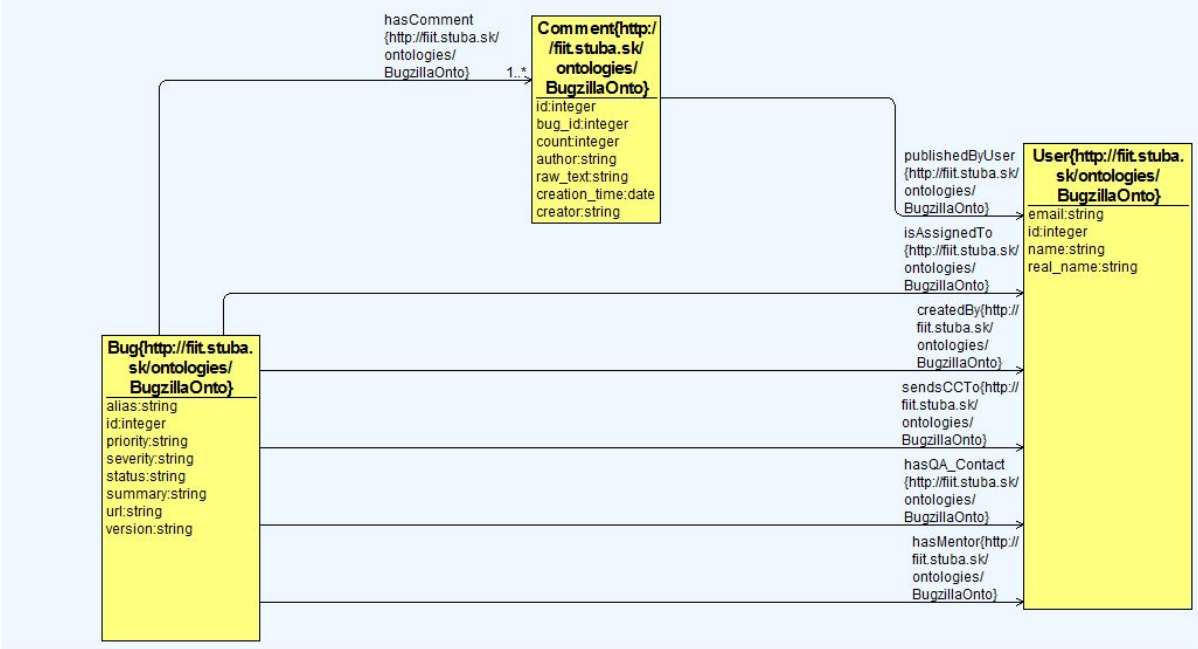
Ontológia pre Git



Ontológia pre Gerrit



Ontológia pre Bugzilla



Implementácia TripleStore

Pre TripleStore je v projekte vytvorený vlastný Solution, v ktorom sú organizované všetky podstatné komponenty. Solution obsahuje niekoľko controllerov, ktoré zastrešujú prácu s trojicami reprezentujúcimi viacero typov – Job a Repository.

- RdfGraphController – poskytuje API pre klienta na vkladanie grafov s IGraph trojicami. Umožňuje ukladať dáta do databázy Jena a tiež z nej dáta získavať. Nepracuje priamo so samostatnými trojicami, ale s celými grafmi, do ktorých sa vložia potrebné trojice. Prichádzajúce trojice sa do RDF databázy ukladajú do grafov, ktoré sú špecifické pre konkrétne dvojice projekt-repozitár. To umožňuje v prípade potreby jednoducho odstrániť záznamy pre konkrétny repozitár vybraného projektu.

- RepositoryController – obsahuje základné funkcie pre prácu s repozitármi v databázi Jena. Umožňuje vytvoriť trojicu s repozitárom, uložiť ju do databázy a ďalšie funkcie.

- JobController - obsahuje základné funkcie pre prácu s jobmi v databáze Jena. Umožňuje vytvoriť trojicu s jobom a iné.

- TripleController – Pomocný controller, ktorý pôvodne obsahoval funkcie na vkladanie grafov do databáze, neskôr sa však táto funkcionalita presunula do RdfGraphControlleru a TripleController v aktuálnom stave projektu neplní

žiadne funkcie.

Spomenuté controllery využívajú množstvo pomocných funkcií na overenie, či zadaný job/repozitár existuje, validácia vstupného grafu podľa ontológií a podobne. Tieto funkcie sú organizované v tzv. StorageHelpers, čo sú triedy s pomocnými funkciami pre contrellery, connectory a iné.

- FusekiStorageFactory – Obsahuje základnú funkcionalitu pre vytvorenie connectora do použitej databázy.
- InMemoryStorageFactory – Obsahuje základnú funckionalitu pre vytvorenie dočasného RDF uložiska v pamäti
- IStorageFactory – Abstraktná trieda, od ktorej dedia všetky typy connectorov
- JobHelper – Obsahuje pomocné funkcie pre prácu s trojicami jobov a IGraphom. Umožňuje overiť, či daný job existuje v databáze alebo vloží zadané joby do grafu.
- RepositoryHelper - Obsahuje pomocné funkcie pre prácu s trojicami repozitárov. Umožňuje overiť, či daný repozitár existuje v databáze alebo vloží zadané repozitáre do grafu.
- NameSpaceHelper – Obsahuje základne funkcie pre prácu s prefixami, ktoré sú neoddeliteľnou súčasťou trojíc
- TripleHelper – Obsahuje funkcie pre načítanie ontológií a validáciu vstupného grafu proti týmto ontológiam.

1.4.2 Balancer

Na serveri sme potrebovali vyhradiť časť, ktorá bude spravovať jednotlivých klientov, joby a zároveň bude oddelená od webovej časti a databázy Jeny.

Rozhodli sme sa preto v rámci servera implementovať ďalší projekt, ktorý bude zodpovedný za časti spomenuté vyššie, projekt sme nazvali Balancer.

Balancer je tak ako aj zbytok servera implementovaný v jazyku c# pod podporou asp.Net.

Samotný balancer sa rozdeľuje na dva základné časti.

ClientController

Balík je zodpovedný za komunikáciu medzi serverom a klientami. Komunikácia medzi týmito dvoma komponentamy prebieha cez HTTP posts a HTTP gets.

Vďaka tejto implementácii je klient schopný dopytovať si z databázy len určité dáta.

Kontroler prideľuje joby jednotlivým klientom a finalizuje dokončené joby. Poprípade

sa stará o špeciálne stavy, ktoré nastali.
Napríklad, klient nedosťahoval celý job a už nemôže viacej sťahovať z dôvodu obemdzenia API. V takom prípade rozdelí kontroler daný job na dva. Jeden z nich je dokončený (práca ktorú už klient vykonal) a druhý je úplne nový. Nový job sa následne, môže prideliť inému klientovi.

Strategies

Balík je zodpovedný za vytváranie jobov podľa špecifikovaných stratégií v type repozitára.

V schéme databázy vidíme, že každý repositoryType má v sebe stĺpec strategy. Ten nám hovorí akú stratégiu využiť pri vytváraní/ spravovaní jobov daného repozitáru.

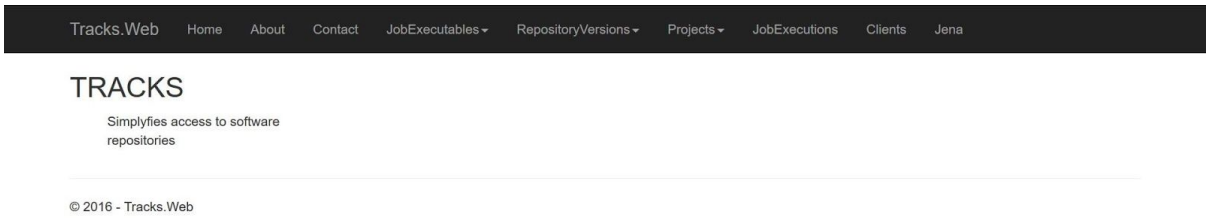
Každý typ repozitáru by mohol mať stanovený iný spôsob sťahovania dát. Napríklad pri Stackoverflow je to podľa dátumu. Na základe tejto skutočnosti treba aj upraviť to ako bude samotný job vyzerať a ako sa job vytvorí. Práve tieto problémy riešia stratégie v tomto balíku.

1.4.3 Web

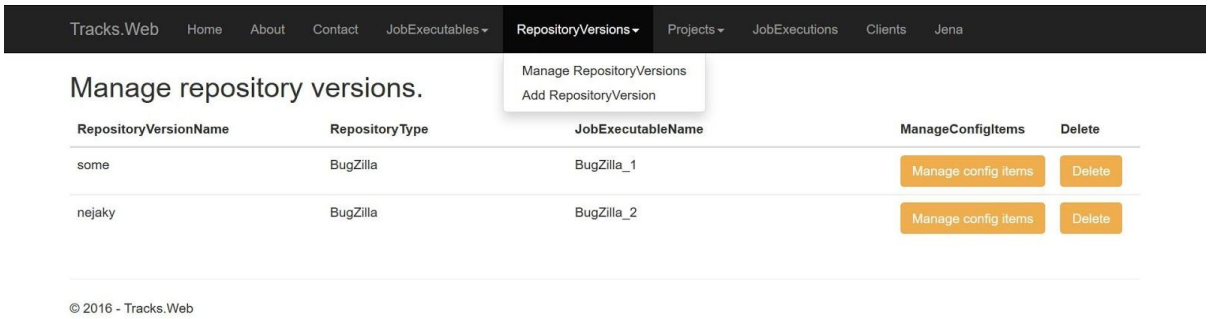
Webové rozhranie je realizované prostredníctvom technológie ASP.NETCore a Razor. ASP.NET Core je moderná technológia nezávislá od platformy založená na tvorenie webových aplikácii. ASP.NET Core aplikácie môžu bežať pod .NET Core alebo pod plnou verziou .NET frameworku. Aplikácia, ktorá bola vyvíjaná môže bežať na rôznych platformách ako Windows, Mac a Linux.

ASP.NET Core MVC je framework pre tvorenie webových aplikácii a jednotlivých API využívajúci návrhový vzor Model-View-Controller. Tento framework je jednoduchý open source, ktorý sa dá jednoducho testovať.

Razor je značkovací jazyk pre zakotvenie serverového kódu na webové stránky. Samotný razor pozostáva zo značiek Razoru, C# a HTML. Súbory, ktoré obsahujú Razor majú príponu *.cshtml*



Obr. 1. Úvodná obrazovka aplikácie



Obr 2. Ukážka práce s aplikáciou

1.4.4 SQL Databáza

Dáta o repozitároch, ktoré treba stiahnuť, používateľ môže zadávať pomocou webového rozhrania. Samotné webové rozhranie musí tieto dáta efektívne ukladať. Pre ukladanie takýchto dát sme zvolili SQL databázu.

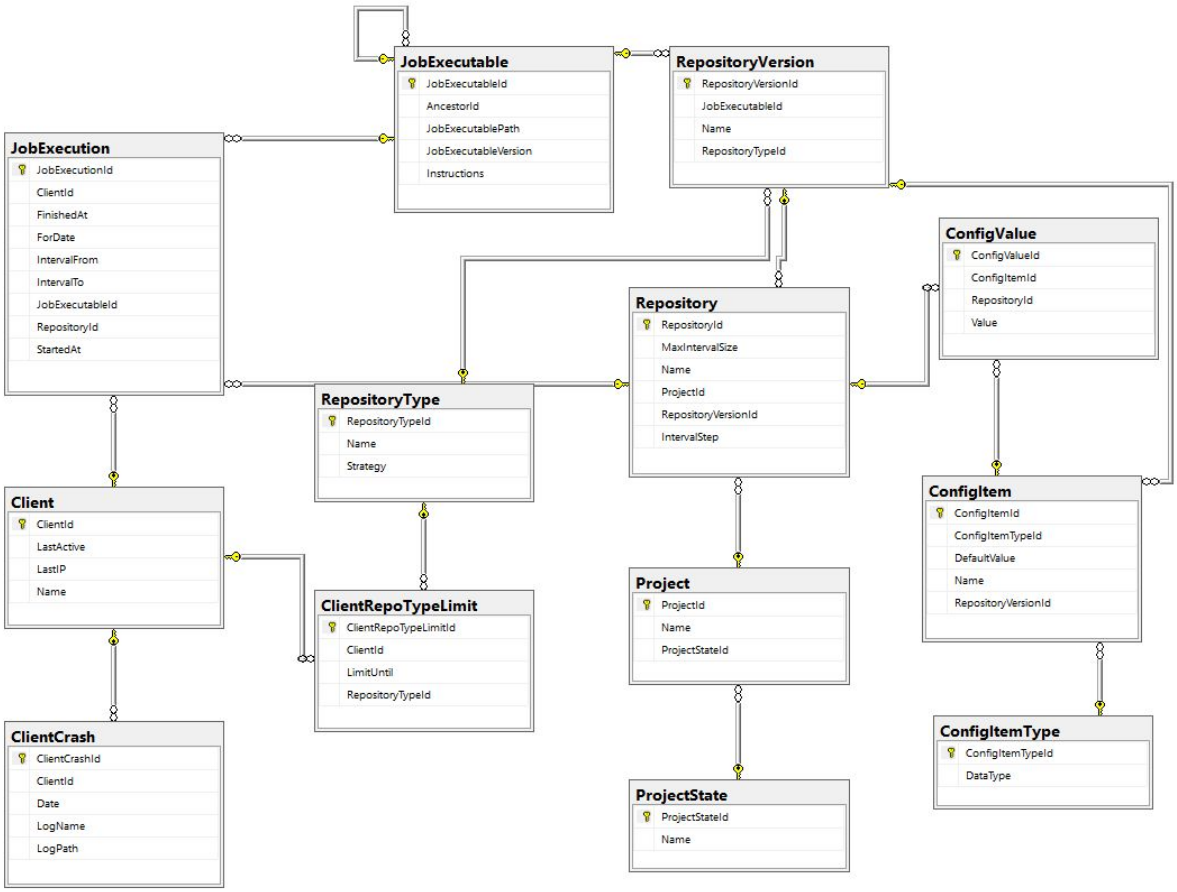
Databáza je implementovaná pomocou MSSQL servera. Vytvára sa pomocou snipetu Code-First. Ten funguje tak, že pri spustení webového servera sa overí, či je databáza vytvorená ak nie je tak sa vygeneruje. V prípade, že je ale má inú schému ako aktuálna Code-First databáza uložená na webovom serveri, tak sa pomocou migrácií databáza na MSSQL serveri aktualizuje podľa definovanej schémy.

K databáze sa pristupuje pomocou Entity frameworku 7 výhradne len z webového servera. Ak klienti vyžadujú nejaké dáta z MSSQL databázy tak si ich vypýtajú cez http dopyty skrz webový server. Entity framework funguje ako ORM, podporuje SQL, Entity-SQL (konkurencia HQL) a LINQ. Zakomponovali sme ho z dôvodu ľahšej práce s DB a zároveň ako prostredníka ku pripojeniam k DB (má plnú podporu Database connection poolingu).

Schéma DB

- *ProjectState* – Obsahuje možné stavy aké môže projekt nadobudnúť.
    - Opis stĺpcov
        - Name (String): Názov stavu projektu.
          New, active, closed
- *Project* – Obsahuje jednotlivé projekty vytvorené používateľom.
    - Opis stĺpcov
        - Name (String): Názov projektu ako celku.
          Eclipse
- *Repository* – Obsahuje konkrétne repozitáre určené na stiahnutie k danému projektu.
    - Opis stĺpcov
        - Name (String): Názov repozitáru.
          EclipseGitHub
        - MaxIntervalSize (int): Po koľkých % sa bude deliť jobExecutable.
          50
- *RepositoryType* – Obsahuje typy repozitárov, ktoré dokážeme sťahovať.
    - Opis stĺpcov:
        - Name (String): Názov typu repozitáru.
          GitHub
        - Strategy (String): Typ stratégie, ktorá sa má využiť pri rozbíjaní daného typu repozitáru.
          JobExeStrategyByDay
- *RepositoryVersion* – Obsahuje verzie repozitárov, ktoré dokážeme sťahovať.
    - Opis stĺpcov:
        - Name (String): Názov konkrétnej verzie typu repozitáru.
          Github version 1
- *ConfigItemType* – Obsahuje dátové typy, ktoré používateľ využíva pri zadávaní configItemov.
    - Opis stĺpcov:
        - DataType (String): Typ premennej.
          String
- *ConfigItem* – Obsahuje parametre, ktoré je možné nastavovať pre daný repositoryVersion.
    - Opis stĺpcov:
        - Name (String): Názov prametra, ktorý je možné nastaviť.
          URL
        - DefaultValue (String): Základná hodnota, ktorá sa predvyplní.
          www.whatever.com
- *ConfigValue* – Obsahuje konkrétne hodnoty jednotlivých parametrov.
    - Opis stĺpcov:
        - Value (String): Konkrétna hodnota parametra
          https://github.com/eclipse/eclipse
- *JobExecutable* – Obsahuje informácie o DLL, ktoré slúžia na sťahovanie jednotlivých typov repozitárov.

    - Opis stĺpcov:
        - JobExecutablePath (String): Fyzická cesta k DLL na servery. C://Path/
        - JobExecutableVersion (String): Názov DLL github_1
- *JobExecution* – Obsahuje joby rozparsované pomocou stratégie určenej v repositoryType. Práve tieto joby sťahujú jednotlivý daemoni.
    - Opis stĺpcov:
        - ForDate (DateTime): Dátum, pre ktorý je job vytvorený.
        - IntervalFrom (int): Od akého intervalu sa sťahujú dáta pre daný deň (inclusive). %
        - IntervalTo (int): Po aký interval sa sťahujú dáta pre daný deň (exclusive). %
        - StartedAt (DateTime): Kedy bol daný job pridelený klientovy.
        - FinishedAt (DateTime): kedy bol daný job ukončený.
- *Client* – Obsahuje, konkrétnych daemonov, ktorý sťahujú dáta.
    - Opis stĺpcov:
        - Name (String): Názov klienta, ktorý používateľ zadal. Lukáš
        - LastIp (String): Posledná IP z ktorej klient pristúpil na server. 171.151.181.9
        - LastActive (DateTime): Kedy bol klient naposledny online.
- ClientCrash - Obsahuje informácie o pádoch daemona.
    - Opis stĺpcov:
        - Date (DateTime) : Dátum kedy bol log pirjatý.
        - LogName: Názov logu.
        - LogPath: Cesta k logu
- ClientRepoTypeLimit - Záznam dokedy má kliet ban na sťahovanie na určitom type repozitáru.
    - Opis stĺpcov:
        - LimitUntil (DateTime) - Dátum dokedy má klient ban.



Príklad: vytvorenie nového projektu s jedným repozitárom na sťahovanie pomocou webového rozhrania

Predpoklady

- Inicializované tabuľky ConfigItemType, ProjectState, RepositoryType.
- Mať vytvoreného deamona a jeho záznam v tabuľke Client.
- Mať vytvorené DLL na sťahovanie danej verzie repozitára.

Používateľ najprv vytvorí záznam v tabuľke RepositoryVersion, pridelí mu vytvorené DLL (nový záznam v JobExecutable), RepositoryType a parametre, ktoré daný repozitár vyžaduje (nové záznamy v tabuľke ConfigItem).
Následne vytvorí záznam v tabuľke Project, tomu sa automaticky pridelí stav „new“ z tabuľky ProjectState. Projektu pridelí nový Repository, stanový mu jeho verziu a vyplní parametre daného RepozitáruVersion.
Po spustení projektu sa vytvoria záznamy v JobExecutions, ktoré sa následne prideľujú jednotlivým klientom.

1.5 Klient

Podstatnou časťou ekosystému TRACKS je klientská aplikácia - tiež nazývaná aj “deamon”,

pretože na pozadí vykonáva serverom pridelené činnosti. Existencia klientských aplikácií v systéme vyplynula z problému sťahovania veľkých repozitárov centralizovanie - server by udelil sťahujúcemu zariadeniu ban , kvôli vysokému počtu requestov z jedného zariadenia.

Vzhľadom na tento fakt bola architektúra TRACKS navrhnutá tak, aby pri sťahovaní veľkých repozitárov sťahovaný repozitár distribuovala medzi klientské aplikácie, ktoré stiahnu a spracujú repozitár lokálne a následne výsledky odošlú na TRACKS server. Nasledovná podkapitola opisuje vývoj klientskej aplikácie TracksDeamon.

1.5.1 Analýza

Cieľovou platformou pre klientské aplikácie je Windows.. Výber platformy teda ovplyvnil aj výber technológie - použitá technológia na implementáciu klienta je C#. Pre potreby projektu bolo potrebné nájsť stále podporovanú knižnicu pre C#, ktorá umožňuje v klientskej aplikácii vytvoriť notifikačnú ikonu a tzv. vyskakovacie okno.

1. **Windows Forms**

Výhody
- možnosť notifikačnej ikony
- Priama podpora v .NET frameworku

Nevýhody
- Nie je umožnené vytvoriť si vyskakovacie okno s vlastným GUI

2. **Harcodet WPF NotifyIcon**

Knižnica implementovaná pomocou WPF (Windows Presentation Foundation) slúžiaca na vytvorenie vlastnej notifikačnej ikony a vyskakovacieho okna.

Výhody
- Možnosť vytvorenia notifikačnej ikony
- Možnosť vytvorenia vyskakovacieho okna s vlastným GUI
- Stále podporovaná (posledný release Version 1.0.8 - April 2nd 2016)

Nevýhody
- Potreba dodatočného NuGet balíka

Výsledok analýzy

Pre project TRACKS bola vybraná na implementovanie klientskej aplikácie knižnica Hardcodet WPF NotifyIcon. Windows Forms síce podporuje notifikačnú ikonu v trayi, ale je pomocou nej možné naimplementovať pop-up tray okno, preto sa od tejto možnosti ustúpilo.

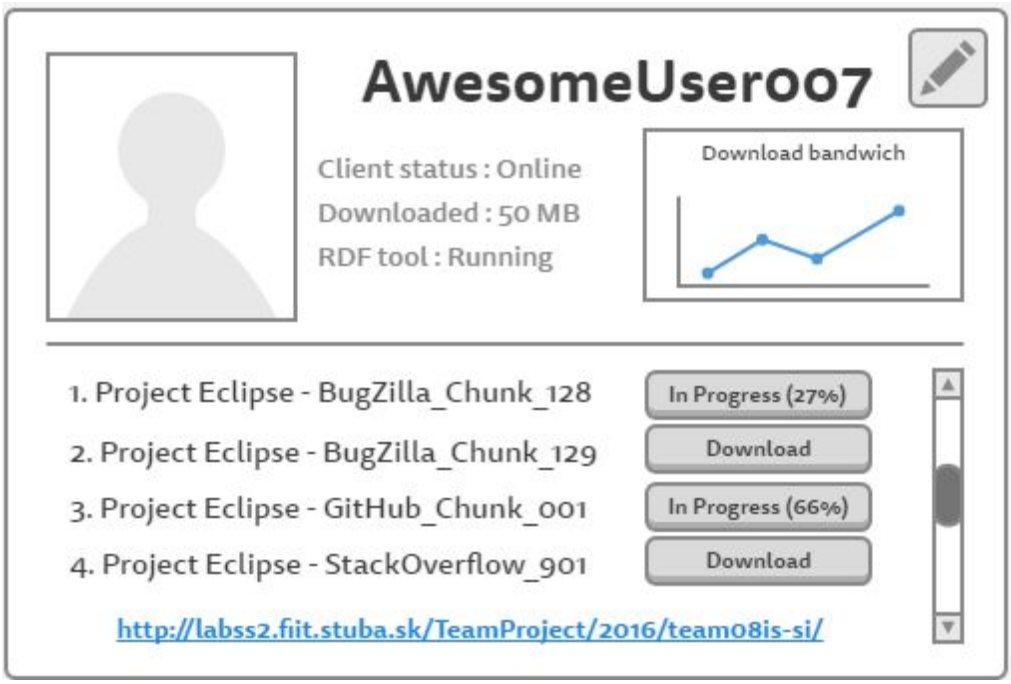
1.5.2 Návrh

Pri návrhu aplikácie sa na začiatku navrhol základný dizajn vyskakovacieho okna, pričom bolo potrebné aby obsahoval základné informácie o behu aplikácie.

- Hlavná obrazovka obsahujúca rýchly prehľad toho, čo práve klient vykonáva
- Dodatočná obrazovka s nastaveniami

- Možnosť nastavenia mena klienta.
- Možnosť skrytia aplikácie do notifikačnej lišty po zatvorení aplikácie
- List obsahujúci prehľad sťahovaných repozitárov

WireFrame obsahujúci základný náčrt GUI klienta



Vo fáze návrhu sme tiež vybrali knižnice, ktoré bude aplikácia používať :

- Logovanie - SimpleLog utilita umožňujúca viacúrovňové logovanie
- Automatické updatovanie - Clickonce knižnica umožňujúca jednoduchú kontinuálnu distribúciu aktuálnej verzie aplikácie klientovi
- Na komunikáciu so serverom bude použitý HttpClient
- Ikony voľne dostupné na internete pod akceptovateľnou licenciou (zatiaľ je postačujúca aj “free for non-comercial use” licencia)
- Farebnú tému tímu - červenú, neskôr sa ale prešlo na modrú farebnú tému

1.5.3 Implementácia

V nasledovnej časti je popísaná implementácia klientskej aplikácie.

Funkcionalita

Aplikácia zahŕňa viaceré triedy, ktoré obsahujú jednotlivé logicky zoskupené časti funkcionality. Hlavná funkcionalita aplikácie je naimplementovaná v triede MainWindow,ktorá ovláda celý životný cyklus aplikácie. Nachádzajú sa tu vytvorené metódy pre spracovanie callbackov GUI a následne sú z tejto triedy volané ostatné časti aplikácie.

V klientovi sa ďalej nachádzajú pomocné triedy pre mapovanie dát jobov do UI klienta.

Trieda DownloadListItem používa návrhový vzor MVVM, ktorý zabezpečuje, že po nastavení OnPropertyChange callbacku bude automaticky sama updatovať svoj stav v UI komponentoch, ktoré ju používajú. Tým je vlastne vývojár odbremenený od updatovania hodnôt v explicitne. Trieda User zoskupuje dáta používateľa vrátane jeho mena, cliendID a aktuálnej IP adresy.

Aplikácia tiež zahŕňa viaceré submoduly, ktoré obsahujú funkcionalitu nezávisle vyvíjanú od klientskej aplikácie.

V súčasnej verzie aplikácie je dokončená nasledovná funkcionalita :

- Implementácia notifikačného okna pomocou WPF NotifyIcon
- Logovanie pomocou SimpleLog loggera
- Nastavenie mena používateľa
- Ohlasovanie na klienta na server
- Pridelenie ID serverom a následné zobrazenie ID v klientovi
- Výber sťahovaného repozitára
- Pridelenie JobExecution a JobExecutable klientu serverom a zobrazenie pridelených úloh v notifikačnom okne
- Spustenie / Zastavenie sťahovania
- Updatovanie stavu sťahovaného jobu priamo do UI aplikácie

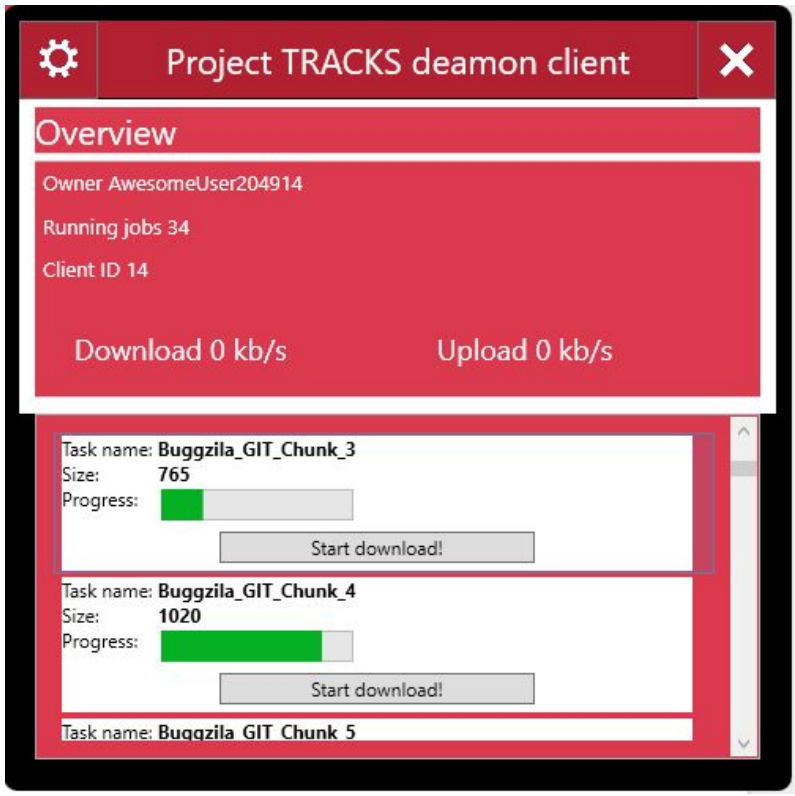
Grafické rozhranie klienta

V súčasnej verzii klienta je nasledovné grafické rozhranie s použitými farebnými motívmi tímu Blank. Do klienta bola tiež pridaná dodatočná obrazovka s nastaveniami, v ktorej môže používateľ meniť svoje meno pod ktorým bude figurovať v systéme.

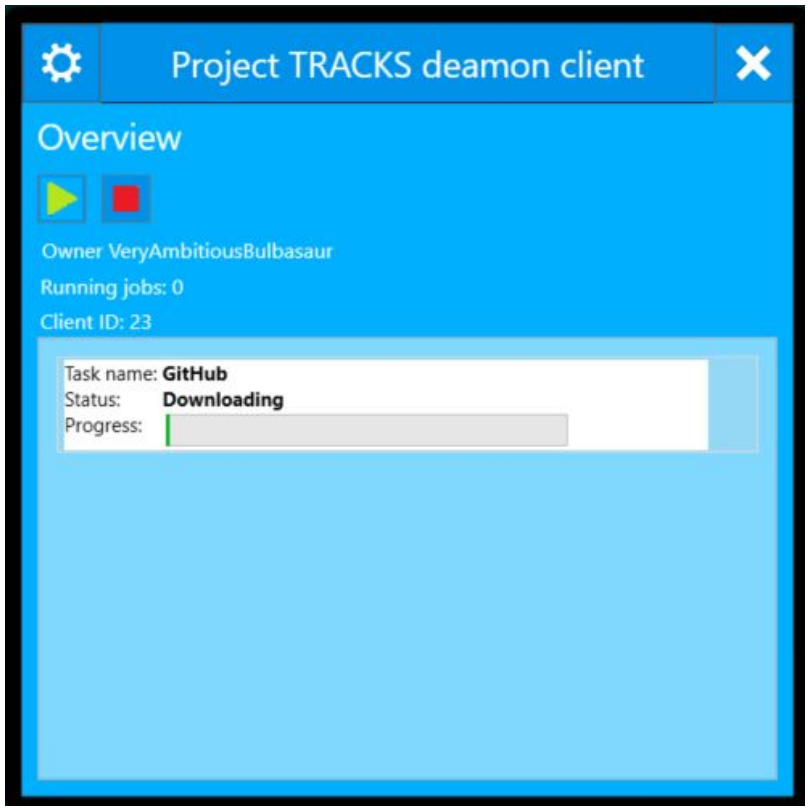
Nasadzovanie

Na účely kontinuálneho nasadzovania aplikácie je použitá knižnica ClickOnce. O sprístupnenie novej verzie klienta online sa stará server TRACKS.

Používateľská príručka klientskej aplikácie sa nachádza v časti prílohy.



Grafické rozhranie klienta, použitá tímová farba - červená (ZS 2016/17)



Nové grafické rozhranie klienta, zvolená farebná téma - modrá (LS 2016/17)

1.6 Stack Overflow Executable

Analýza
Na Stack Overflow existuje verejné Rest API, ktoré nám umožní sťahovať otázky postupne a so všetkým čo potrebujeme. API má možnosť nastaviť si na sťahovanie iba údaje ktoré chceme v rámci šetrenia bandwidthu. Rest API má limity 300 requestov za deň na IP adresu a 30 requestov za sekundu.
Toto API podporuje sťahovanie zo všetkých stránok ktoré sú typu Stack Overflow.

Návrh
Ideálne použitie by bolo vyrobiť si filter pri spustení, podľa ktorého by sa sťahovali nastavené stĺpce a nič viacej. Po stiahnutí sa z týchto dát vyrobia trojice podľa ontológie. Po stiahnutí všetkých dát (alebo narazení na limit) sa trojice odošlú na server.

Implementácia
V údajoch prijatých od serveru cez klienta sú dátumy, ktoré treba sťahovať, tag ktorý chceme sťahovať, aj stĺpce. Tie sú prijaté formou dictionary(string, list<object>).
Taktiež prijímame informácie o intervale, ktorý je pridelený jednému klientovi na sťahovanie, ktorý je formou od do.
Každé DLL si vypočíta podľa intervalu koľko z každého dátumu špecifikovaného ma stiahnuť. Vďaky tomu sa jedny údaje nestiahnu viackrát.
Sťahovanie sa deje za pomoci Stacman projektu, ktorého rozhranie špecifikované pre kontakt so stack overflow API.
Odpoveď zo Stacmana prichádza serializovaná na objekty.
Stiahnuté údaje rozbije na trojice presne podľa ontológie a dá informáciu klientovi ako pokročil so sťahovaním v percentách. Keď dosťahuje a dospracováva všetky, odošle ich na server.
Sťahovacie metódy sú implementované asynchrónne, takže nezahltia thread keď čakajú na odpoveď od stacku.
Všetky chyby sú ošetrené pomocou try catch blokov a výpisov.
Ako všetky executables, komunikácia s klientom je riešená pomocou eventov. Klient pri spustení executable subscribne na event executable ako aj executable subscribne na event klienta. Configurácia sa posiela z klienta do executable a počas behu executable oznamuje klientovi priebeh sťahovania.

StackDLL umožňuje sťahovanie stránok otázok alebo používateľov, defaultne podla dátumu vytvorenia a tagov s nimi asociovaných.

Testovanie
Sťahovanie sa testuje a overuje manuálne za pomoci DLLTestera, ktorý umožňuje nezávislé spúšťanie DLL.

Konfigurácia
Stack config môže obsahovať niekoľko druhov vecí:

`stringy include a exclude` ktoré môžu obsahovať všetky možnosti z lokácie https://api.stackexchange.com/docs/questions

`string tag` - jednotlivé hodnoty tagov

`bool testing - true or false` - určené na testovanie odosielania a komunikácie aj keď je limit prejdený

`string downloadSubject - "question" alebo "user"`, podľa toho čo je ciel sťahovania. Question je nastavený defaultne. Question downloads questions posted on specific day, user downloads user created on specific days.

`string serverURL - url` kam poslať trojice. defaultne nastavené na http://team08-16.studenti.fiit.stuba.sk:8008 (nie je potrebné špecifikovať). Ďalšie možnosti: http://localhost:62345

Odporúčaný set configov:

`string "include" = question.answers, question.comments, question.body, answer.body, comment.body, answer.comments`

`bool "testing" = true`
#ak testujeme :)

`string serverURL = http://localhost:62345`
#ak testujeme s lokálnym serverom

1.7 GitHub Executable

Analýza
Množstvo programátorov využíva technológiu Git pre spravovanie zdrojového kódu. Ide o nástroj pre verziovanie. Webová služba, ktorá podporuje vývoj softvéru za pomoci tohto nástroja sa nazýva GitHub. Pokiaľ má programátor bezplatné konto, všetky jeho repozitáre sú verejne dostupné. Platené konto umožňuje používanie súkromných repozitárov. GitHub vznikol v roku 2008 a v súčasnosti prevádzkuje niekoľko desiatok miliónov repozitárov.
Po každom vytvorení repozitára používateľom, sa všetky jeho zmeny v kóde ukladajú ako commity. V zázname o commite, je uložená informácia o tom, kto túto zmenu vykonal, kedy ju vykonal, čo bolo jej obsahom a množstvo ďalších informácií. Práve spracovanie dát o týchto zmenách v kóde prinesú prídavnú hodnotu, ako pre používateľa samotného, tak aj pre celý tím.

Návrh
Pre sťahovanie dát z GitHubu je vytvorený nástroj API GitHub Developer (https://developer.github.com/v3/), ktorý pomocou http dopytov, vie vrátiť všetky údaje o repozitároch, commitoch, používateľoch a podobne. Naším cieľom bude pomocou dopytov opísaných na stránke API získavať všetky údaje o commitoch vykonaných nad jednotlivými repozitármi.

Implementácia
Dopyty, pomocou ktorých vieme získať údaje o commitoch sú opísané na stránke: https://developer.github.com/v3/repos/commits.
Príklady použitia dopytov:

1. Získanie údajov o commite s číslom commitu cislo_commitu, v repozitári nazov_repozitara, ktorý založil používateľ meno_vlastnika.
   http://api.github.com/repos/**meno_vlastnika**/**nazov_repozitaru**/commits/**cislo_commitu**/
   Príklad:
   - http://api.github.com/repos/mapbox/mapbox-gl-native/commits/b49c5a711a0e7a866c2e9fd6c88eb19a77a6a018/
   - http://api.github.com/repos/mapbox/mapbox-gl-native/commits/9eb7f88b2c292d322a104c4580c3ef29958b628b
2. Získanie všetkých commitov z obdobia from - to, v repozitári nazov_repozitara, ktorý vytvoril používateľ meno_vlastnika.
   http://api.github.com/repos/**meno_vlastnika**/**nazov_repozitaru**/commits?since=**from**$until=**to**/
   Príklad:
   http://api.github.com/repos/mapbox/mapbox-gl-native/commits?since=2016-11-27T16:50:24Z&until=2016-11-28T12:39:52Z/

   Príklad (stiahnutie commitov zo dňa 27.11.2016) :
   ```
   DateTime startDate = new DateTime(2016, 11, 27);
   ```

```
DateTime endDate = new DateTime(2016, 11, 28);

String startDateStr = (startDate.ToString("s") + "Z");
String endDateStr = (endDate.ToString("s") + "Z");
```

3. Získanie všetkých tagov repozitáru:
   https://api.github.com/repos/**user_name**/**repos_name**/tags
   Príklad:
   https://api.github.com/repos/eclipsesource/tabris-js/tags

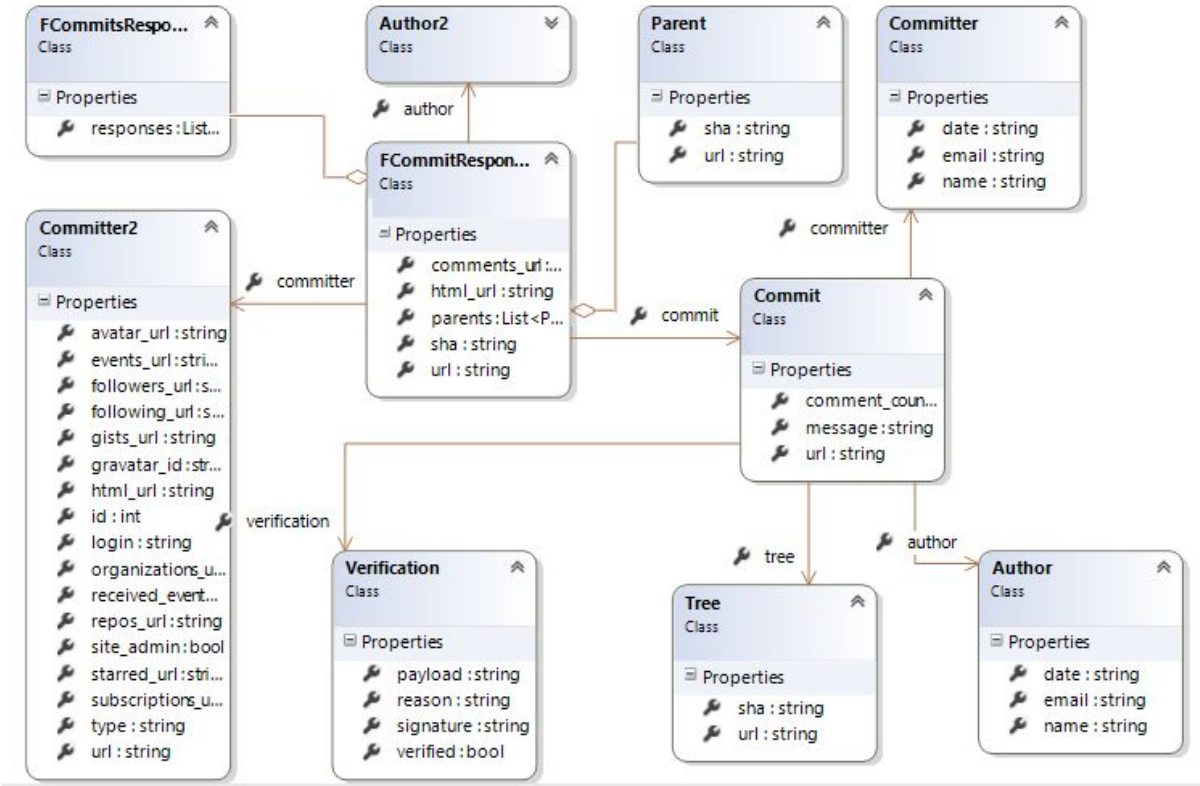
4. Získanie všetkých branches repozitára:
   https://api.github.com/repos/**user_name**/**repos_name**/branches
   Príklad:
   https://api.github.com/repos/eclipsesource/tabris-js/branches

Stratégia sťahovania:

Poznáme meno **organizácie** a **repozitáru, dátum** a **interval**:

1. Vyhľadanie všetkých **branches** pre repozitár.
   a. Vyhľadania všetkých commitov pre všetky branches.
   b. Vytvorenie RDF trojíc o repozitár - branch.
   c. Vytvorenie RDF trojíc o branch - commit.
2. Vyhľadanie všetkých **tagov** pre repozitár.
   a. Vyhľadanie všetkých commitov pre všetky tagy.
   b. Vytvorenie RDF trojíc o repozitár - tag.
   c. Vytvorenie RDF trojíc o tag - commit.
3. Vyhľadanie všetkých **commitov** pre repozitár.
   a. Vytvorenie RDF trojíc o commitoch.
   b. Vyhľadanie všetkých **blobov** pre každý commit.
   c. Vytvorenie RDF trojíc o commit - blob.

Diagram tried zobrazuje odpoveď API na volaný dopyt o informácie o commite.



U klienta sa http odpovede spracovávajú na list objektov. Pre každý záznam z listu sa vytvoria trojice podľa ontológie v RDF formáte. Následne sa trojice do grafu a posielajú balancerovi, ktorý ich kontroluje podľa pravidiel ontológie a posiela na Triple Store. Implementáciu opisuje technická dokumentácia, ktorá je súčasťou prílohy.

Testovanie
Pre testovanie tejto časti projektu sme využívali projekt DLLTester, ktorý náš projekt zbuildoval a spustil. Pre spustenie sme nastavovali konfiguráciu. Návod vyzerá takto:

Git
Config musi obsahovat tieto atributy:

-- string orgName - názov organizácie, pre ktorú chceme sťahovať
-- string reposName - názov repozitára organizácie s názvom orgName
-- bool testing - true or false - urcene na testovanie odosielania a komunikacie aj

ked je limit prejdeny
-- serverURL - adresa servera, kam sa maju odosielat RDF trojice

Príklad nastavenia config:

-- string "orgName" = bugzilla
#alebo: Microsoft

-- string "reposName" = bugzilla
#alebo ak orgName == Microsoft: microsoft.github.io

-- bool "testing" = true
#ak testujeme

-- string serverURL = http://team08-16.studenti.fiit.stuba.sk:8008

Pôvodne sme vyhľadávali všetky comity pre jedného používateľa. Neskôr sme sa rozhodli, že budeme vyhľadávať ionformácie o commitoch pre organizáciu a jej vybraný repozitár. V návode pre vytvorenie konfigurácie je použitá organizácia Microsoft a Bugzilla, a to z toho dôvodu, že mali dostatočný počet tagov, vetiev a komitov. Organizáciu Bugzilla sme testovali preto, lebo sme ju vyhodnotili ako organizáciu, ktorá využíva StackOverflow a zároveň systém Bugzilla.

Po prvotnom testovaní pomocou projektu DLLTester, sme prostredníctvom projektu ZipperTeam08.exe skomprimovali do zip súboru všetky vytvorené dll súbory potrebné pre sťahovanie úloh.

1.8 Bugzilla Executable

Analýza
Bugzilla je bežne používaný komerčný systém pre manažment chýb, ktorý využíva nespočet projektov po celom svete. Analyzovali sme dostupné možnosti sťahovania dát z Bugzilly prostredníctvom Bugzillou podporovaných webových služieb. Bugzilla servery podporujú viacero metód komunikácie s Bugzillou, medzi ktoré patria napríklad REST api a xml-rpc.net api. Odporúčaným spôsobom je REST api, ktoré má v blízkej budúcnosti nahradiť ostatné, no v súčasnosti ho podporuje iba vybraná skupina Bugzilla serverov. Komunikáciou s Bugzilla servermi sa zaoberali mnohé projekty, z ktorých niekoľko je voľne dostupných na online repozitároch. Všetky však s Bugzillou komunikujú prostredníctvom xml-rpc.api, ktoré kedysi postačovalo, no dnes už väčšinou Bugzilla serverov nie je podporované.

Návrh
Vzhľadom na výsledky analýzy sme sa rozhodli komunikovať s Bugzillou pomocou REST api. Na implementáciu sťahovania používame knižnicu asp.net, ktorá sprostredkuje komunikáciu s REST api. Pomocou funkcií tejto knižnice sa vykonávajú jednotlivé dopyty na REST api Bugzilly. V súčasnosti má podporu pre REST api iba obmedzená množina Bugzilla serverov. Medzi ne patrí napríklad http://mozilla.bugzilla.org, z ktorého sme sa rozhodli sťahovať.

Implementácia
Pre vykonávanie dopytov na Bugzilla REST api je potrebné mať autorizačný token, ktorý je následne pridaný na záver dopytov. Autorizačný token možno získať z účtu Bugzilly. Použiteľný autorizačný token musí byť obsiahnutý v konfiguračnom súbore pre Bugzilla executable.

Dáta z bugzilly sa sťahujú z jedného dňa po krokoch. Kroky sú hodnoty v intervale od 1 po 100, ktoré určujú percentuálnu časť dňa, z ktorej sa majú stiahnuť informácie o bugoch. Príklad: interval 75 - 80 v prepočte na hodiny a minúty znamená 18:00 - 19:12. Samotné sťahovanie je implementované http dopytmi pomocou funkcií ASP.NET API. Vytvorenie spojenia vyzerá nasledovne:

```
client = new HttpClient();
HttpResponseMessage response;
client.BaseAddress = new Uri(targetServerAddress);
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(new
     MediaTypeWithQualityHeaderValue("application/json"));
```

Po nastavení http klienta môžeme vykonávať http dopyty na zvolený server, v tomto prípade *http://bugzilla.mozilla.org/rest/.* Vykonanie dopytu na stiahnutie jedného bugu z intervalu vidieť nižšie:

```
 bugQuery = "bug?creation_time=" + startingDate + startingTime +
"Z&&limit=1&&offset=" + offset;

response = await client.GetAsync(bugQuery + "&&token=" +
authorizationToken);
```

Týmto spôsobom sa stiahnu všetky bugy (a ak je to v konfigurácii nastavené, aj ich príslušné komentáre), ktoré spadajú do intervalu. Tieto bugy sa následne uložia do RDF trojíc a odošlú na server.

Konfigurácia
Pre správnu funkčnosť Bugzilla dll je potrebné mu prostredníctvom konfiguračného súboru odoslať niekoľko hodnôt:

serverURL:string
- url servera, na ktorý sa majú odoslať stiahnuté dáta uložené do RDF trojíc
- odporúčaná hodnota: “http://team08-16.studenti.fiit.stuba.sk:8008”
- predvolená hodnota: “http://team08-16.studenti.fiit.stuba.sk:8008”

ForDate:DateTime
- dátum dňa, z ktorého sa majú sťahovať bugy
- odporúčaná hodnota: od balancera
- predvolená hodnota: včerajší dátum (podľa systémového času)

downloadComments:bool
- boolovská hodnota, ktorá dáva dll pokyn, či k bugom sťahovať aj komentáre
- *true* = sťahovať, *false* = nesťahovať
- odporúčaná hodnota: *true*

- predvolená hodnota: *false*

authorizationToken:string
- autorizačný token patriaci konkrétnemu Bugzilla účtu, bez ktorého nie je možné vykonávať niektoré dopyty
- odporúčaná hodnota: autorizačný token ľubovoľného Bugzilla konta
- predvolená hodnota: “h9FtudjflXSbqULTebLno5X1CJh6ARM1ygeln74q”

targetServer:string
- adresa konkrétneho Bugzilla servera, z ktorého majú byť sťahované dáta
- odporúčaná hodnota: od balancera
- predvolená hodnota: “http://bugzilla.mozilla.org/rest/”

1.9 Výroba nových DLL spolupracujúcich s našim klientom a serverom

Je potrebné stiahnuť najnovšiu verziu nášho nuggetu z devacts serveru.
Následne keď sa na ňu odkážeme, implementovaním JobExecution z daného nuggetu budeme mať základnú funkčnú verziu DLL.

JobExecution obsahuje niekoľko metód, ktoré je možné prekonávať, ale ich základná implementácia je potrebná pre správnu komunikáciu s klientom alebo serverom. V prípade potrebných zmien na nich je dobré silno sa inšpirovať ich implementáciou.
Konkrétne ide o metódy:
```
AfterStop()
SendTriples()
```

Pre bežné úpravy v JobExecution existujú virtual metódy určené na to byť prekonávané. Konkrétne ide o metódy:
```
BeforeStart()         - vykoná sa práve raz pred spustením samotnej
funkcionality DLL
AfterStart()          - vykoná sa práve raz po spustení DLL (nie
synchrónne!)
BeforeRestart()             - vykoná sa práve raz pred reštartovaním
DLL (Zatiaľ nevyužitá funkcionalita)
BeforeStop()          - vykoná sa práve raz pred zastavením DLL
BeforeStepProcessed() - vykoná sa pravidelne pred každým vykonaným
krokom sťahovania
AfterStepProcessed()  - vykoná sa pravidelne po každom vykonanom
kroku sťahovania
DownloadStep()  - vykoná sa raz v každom kroku sťahovania. Ide o
metódu samotného sťahovania, ktorá by mala vraciať IGraph s
trojicami. Hlavná funkcionalita DLL sa očakáva práve tu.
```

Config items nastavované na serveri do DLL prichádzajú v premennej `configuration`, ktorá je typu `Dictionary<string, List<object>>`. Pod jedným value sa môže nachádzať viac ako jedna hodnota v zozname podľa potreby. Na začiatku spúšťania DLL je potrebné očakávané config itemy vyparsovať do lokálnych premenných.

V prípade zastavenia vykonávania z dôvodu prekonania limitu na repozitári je potrebné nastaviť flag `stopExecution` na `false` a do `limitTime` premennej nastaviť očakávaný čas kedy už bude sťahovanie opäť možné.
V prípade akéhokoľvek zastavenia je aj potrebné mať nastavenú hodnotu `downloadedStep` na posledný úspešne dokončený krok sťahovania.

V prípade potreby je možné zaimplementovať priamo IJobExecution, ale je potrebné použiť metódy `AfterStop()` a `SendTriples()` z JobExecution pre validnú komunikáciu.

Po výrobe DLL a jeho zbuildovaní je potrebné sťiahnuť zipper zo stránky a zabaliť DLL aj so všetkými referencovanými balíčkami za pomoci tohto zippera.

2 Zoznam príloh

Príručky:
Pouzivatelska_prirucka_klient.docx.pdf
Pouzivatelska_prirucka_server.pdf

Návody:
Apache jena fuseki tutorial - user tutorial.pdf
Api_pre_oznamovanie_clienta_na_server.docx.pdf
Nugety_vytvaranie_aplikovanie.pdf
Rozbehanie_dotnetcore_projektu_-TRACKS_server_-_na_unixe.pdf
Rozbehanie_MSSQL_lokalne.pdf
Rozbehanie_MSSQL_na_serveri.pdf
Server_manual.pdf
test_report.pdf
Tvorba_unit_testov.docx.pdf
vyrábanie_ontológií.pdf
Výroba nových DLL.pdf
Vytvorenie_migracie.pdf
Vytvorenie_user.json.pdf
MetodikaPraceNaprojekte.pdf
Gitworkflow.pdf
Codeconventions.pdf

Pre nastavovanie konfiguračných súborov:
Bugzilla_config.txt.pdf
Git_config.txt.pdf
Stack_config.pdf

2.1 Súbory na CD

Finálna dokumentácia - Tento dokument spolu so všetkými prílohami
JobExecutables - Vybuildovane a DLL a aj kody k nim.
Ontológie - owl a bmp ontológií pre Git, BugZilla, StackOverflow a Gerrit
Projekt - Vybuildovaný server a klient aj kódy k nim, spolu s nugetom
Technická dokumentácia - vygenergovaná pre klienta a server
TP CUP - článok na IITSRC, dotazník a prezentácia na robime.it
www stranka - statická verzia stránky na záver projektu
Zipper - nástroj na zippovanie používaný pre zipovanie DLL
server_manual_videa.zip - inštruktážne videá na používanie servera