Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Rozhranie pre distribuovaný výpočtový systém
Dokumentácia k riadeniu projektu

Vedúci tímu: Ing. Peter Lacko, PhD.
Členovia tímu: Bc. Pavol Čurilla, Bc. Patrik Gallik, Bc. Martin Kaššay,
Bc. Matej Kloska, Bc. Juraj Kochjar, Bc. Roman Roštár
Akademický rok: 2014/2015

Obsah

1 Úvod

Tento dokument slúži ako dokumentácia opisujúca postupy a metódy riadenia tímového projektu použité pri vytváraní webového rozhrania pre distribuovaný výpočtový systém. Cieľom projektu je vytvoriť webové rozhranie, kde bude možné prehľadným a intuitívnym spôsobom poskytnúť širokému spektru fakultných používateľov možnosť vkladať a sledovať svoje výpočtovo náročné výskumné úlohy. Tieto úlohy bude možné vkladať vďaka projektu BOINC@FIIT. Tento projekt využíva zdieľaný výpočtový výkon svojich používateľov, a zároveň jednotlivé úlohy spracuje a rozdistribuuje medzi projektových participantov.

Na to aby bolo možné projekt BOINC@FIIT plnohodnotne využívať, a tak prinášať fakultným výskumníkom z rôznych oblastí výsledky nimi zadaných úloh, je nutné zvýšiť povedomie o tomto projekte a motivovať širokú verejnosť k poskytnutiu svojho často nevyužitého výpočtového výkonu v prospech výskumu na fakulte.

2 Predstavenie tímu

2.1 Opisy členov tímu

2.1.1 Matej Kloska

Počas strednej školy som sa primárne zameriaval na vývoj webových aplikácii a neskôr na vývoj multiplatformových aplikácii v programovacom jazyku Python. Počin vtedajšieho snaženia bol prezentovaný na IIT.SRC 2011 v kategórii Junior publikovaním príspevku s názvom Sincerus: Mutiplatform Python Application Framework. Počas štúdia na fakulte som pracoval a pracujem na projekte AdBOOST. V začiatkoch na pozícii junior developera, počas 3 rokov práce na projekte som sa prepracoval na technického vedúceho projektu. Používané technológie a prog. jazyky: Python, Java, MySQL, MongoDB, Redis, RabbitMQ (spracovanie dávkových distribuovaných úloh), nginx, TornadoServer (fault-tolerant retargeting web services). Google compute engine na dynamické distribuované výpočty, skúsenosti s Amazon AWS. Ďalej zručnosti v oblasti reklamy: Google AdWords, Facebook Ads, Sklik, SEO - Google Analytics, riadenie projektov (JIRA), dokumentácia projektov (Atlassian Confluence), Git, CI, NodeJS, Ruby On Rails.

2.1.2 Patrik Gallik

Dlhšiu dobu pracujem ako developer v Lighting Beetle, kde sa momentálne venujem vývoju frontendu web aplikácii (AngularJS, React), respektíve embednutých desktop aplikácií. Ako firma máme silný dôraz na UX a pri vývoji spolupracujeme s vývojarmi klientskej firmy.
Zaoberám sa taktiež vývojom backend aplikácií, aktuálne pomocou PHP frameworku Laravel. Zoznam technológií a rôznych pomocných nástrojov, ktoré aktívne používam: Javascript, NodeJS, Grunt, PHP, Java, Vagrant, Stash, Jira, Apache Cordova.

2.1.3 Juraj Kochjar

Niekoľko rokov sa venujem vo voľnom čase grafike a dizajnu. Myslím si, že mám schopnosti a skúsenosti, aby som bol schopný navrhnúť a zrealizovať používateľské rozhranie so zreteľom na pravidlá UI/UX. Pri návrhu používam hlavne CorelDRAW. Z frontendu sa nebránim prakticky ničomu novému - mám rád výzvy. Z pohľadu frontendu ma zaujíma jQuery, HTML5, CSS3 a AngularJS. Na strane backendu mám základné skúsenosti s NodeJS. Počas práce na bakalárskom projekte som použil grafovú databázu Neo4j, ako zástupcu NoSQL databáz.

2.1.4 Martin Kaššay

Posledný rok pracujem v Aldobec Technologies na pozícii PHP developer, kde okrem PHP vývoja riešim veci ohľadom jQuery, Javascript-u, MySQL a PgSQL. Aktívne pracujem s Twitter Bootstrap, Jira a Git. Vo voľnom čase však radšej pracujem v C / C++. Bakalársky projekt som vypracoval v C++ s použitím Cuda, takže paralelné programovanie mi nerobí problém.

2.1.5 Roman Roštár

Som nadšenec nových technológií, počas leta som pracoval ako C# Developer vo Visicom a.s. a zároveň som navštevoval technickú časť Startup Summer school v TheSpot.sk, kde som sa zlepšoval ako v oblasti web dev, tak aj v oblasti tvorby startup projektov. Počas kurzu sme pracovali najmä v Ruby on Rails s dôrazom na TDD/BDD. Momentálne sa v rámci startupu, ktorý počas tejto letnej školy vznikol najviac venujem fullstack práci v Ruby on Rails a v blízkej budúcnosti budem intenzívne pracovať na vývoji aplikácii pre platformy Android a iOS.

2.1.6 Pavol Čurilla

Momentálne pracujem ako Java Developer v Gratex International, a. s., pričom využívam najmä Javu, Groovy, mongoDB, JavaScript a Git. Okrem toho som pracoval napríklad s jMonkey, AspectJ, XML a SQL.

2.2 Motivácia

Zadaním je vytvoriť webové rozhranie pre existujúci projekt BOINC@FIIT. Vývoju webu a webovým technológiám sa väčšina z členov tímu aktívne venuje vo voľnom aj profesnom čase. Medzi hlavné dôvody, prečo sme sa rozhodli venovať tejto téme je, že máme skúsenosti aj s paralelným programovaním, čo nám pomôže lepšie pochopiť podstatu systému, pre ktorý navrhujeme rozhranie

Medzi naše nápady, respektíve riešenia, ktorými by sme chceli aktuálny systém vylepšiť, patria:

- prepracovať dizajn výsledného rozhrania s dôrazom na intuitívnosť a poskytnúť prívetivé prostredie pre široké spektrum bežných používateľov, (bez nutnosti ovládania konzolových, či iných príkazov). S tým úzko súvisí zjednodušenie procesu zadávania úlohy do systému,
- vybudovanie bázy znalostí pre jednoduchší štart používania rozhrania pre takých používateľov, ktorí sa nikdy nestretoli s platformou BOINC,
- notifikácie o stave vypočtu úlohy, predbežný čas do konca výpočtu,
- vytvoriť dashboard, kde bude možné v reálnom čase sledovať, čo a kde sa aktuálne počíta - počet počítaných úloh, distribúcie medzi používateľmi, počet aktívnych používateľov (resp. ich úloh). Pomocou detailných grafov, nie ale na úkor prehľadnosti a jednoduchosti,
- motivovať používateľa používať tento systém na počítanie jeho úloch pomocou rebríčkov, bodmi, súťažou, oceneniami,
- kontrola bezpečnosti vkladaných výpočtových úloh na základné chyby / útoky zo strany zadávajúceho ako napríklad pretečenie pamäti (nápad, ktorý sme ponúkli pri našom spoločnom stretnutí).

2.3 Úlohy členov tímu

2.3.1 Manažérske úlohy

Úlohy boli pridelené na základe osobných preferencií a skúsenosti daného člena tímu na prvom spoločnom stretnutí po otvorenej diskusii na danú tému.

Meno člena tímu	Rola v tíme
Bc. Roman Roštár	Vedúci tímu
Bc. Patrik Gallik	Hlavný architekt
Bc. Martin Kaššay	Manažér kvality
Bc. Juraj Kochjar	Manažér dokumentovania
Bc. Pavol Čurilla	Manažér podporných prostriedkov
Bc. Matej Kloska	Manažér vývoja

2.4 Podiel práce

2.4.1 Šprint 1

Zodpovedné osoby	Úloha	Ohodnotenie User Story [hodiny]
Bc. Matej Kloska	Konfigurácia serveru	1,5
Bc. Roman Roštár	Konfigurácia REST	1
Bc. Patrik Gallik	Konfigurácia SPA	0,5
Bc. Juraj Kochjar	Návrh landing page	3,5
Bc. Martin Kaššay, Bc. Pavol Čurilla	Ošetrenie nežiadaných - generovaných používateľov v existujúcom projekte	3,5

2.4.2 Šprint 2

Zodpovedné osoby	Úloha	Ohodnotenie User Story [hodiny]
Bc. Juraj Kochjar, Bc. Martin Kaššay, Bc. Pavol Čurilla, Bc. Patrik Gallik	Poskytnutie informácií o projekte BOINC@FIIT	10
Bc. Roman Roštár, Bc. Matej Kloska	Registrácia do projektu	4,5
Bc. Matej Kloska, Bc. Patrik Gallik	Prihlásenie sa do projektu	5,5

2.4.3 Šprint 3

Zodpovedné osoby	Úloha	Ohodnotenie User Story [hodiny]
Bc. Matej Kloska	Migrácia databázy	3,75
Bc. Roman Roštár	Transakčné e-maily	9
Bc. Juraj Kochjar	SPA - návrhy	9
Bc. Patrik Gallik	SPA - profil a nastavenia	4
Bc. Martin Kaššay	Landing page	3,5
Bc. Roman Roštár, Bc. Patrik Gallik, Bc. Martin Kaššay, Bc. Juraj Kochjar, Bc. Pavol Čurilla, Bc. Matej Kloska	Vypracovanie metodík	N/A
Bc. Roman Roštár, Bc. Patrik Gallik, Bc. Martin Kaššay, Bc. Juraj Kochjar, Bc. Pavol Čurilla, Bc. Matej Kloska	Vypracovanie dokumentácií	N/A

2.5 Manažment komunikácie

Kľúčovou úlohou riadenia akýchkoľvek procesov v tíme je vzájomná, konštruktívna a intenzívna komunikácia členov tímu. Riadená a moderovaná diskusia prispieva k skvalitneniu všetkých procesov v tíme, či už z pohľadu dodržiavania časového, kvantitatívneho a v neposlednom rade kvalitatívneho aspektu projektu. Výsledkom správnej analýzy požiadaviek zadávateľa projektu a následnou spoločnou komunikáciou, je záväzná identifikácia cieľov, ktorá minimalizuje riziko vytvárania nadbytočnej funkcionality, tým pádom efektívne formuje časový horizont dokončenia projektu. Pre náš tím je komunikácia kľúčová aj z hľadiska kvality a kompatibility nami vyvýjaných modulov systému - SPA aplikácie a REST backendu projektu.

2.6 Stretnutia tímu

Oficiálne stretnutia tímu prebiehali každý týždeň vo štvrtok o 10:00. Ich témou bolo zhodnotenie vykonanej práce za predchádajúci týždeň a naplánovanie ďalšej. Okrem toho prebiehali neformálne stretnutia, ktoré sa konali väčšinou po prednáškach a nemuseli sa vždy konať v plnom počte. Ich témou bolo hlavne riešenie menších problémov pri implementácií, prípadne upevňovanie tímovej morálky.

2.7 Komunikačné nástroje

Pri výbere nástrojov sme vychádzali zo skúsenosti a preferencií jednotlivých členov tímu. Analyzovali sme najznámejšie nástroje a dohodli sme sa na nasledujúcich nástrojoch:

HipChat
Chatovací klient od firmy Atlassian, ktorá svojimi softvérovými produktmi zastrešuje podstatnú administratívnu časť projektu. Slúži na rýchle dorozumievanie sa medzi členmi tímu. Vytvorili sme chatovacie miestnosti pre každý repozitár, aby bola komunikácia prehľadnejšia. Výhodou sú notifkácie pri novej správe, ktoré sú odosielané na mail, pokiaľ používateľ nie je v danom momente prihlásený, a takisto možnosť označiť pri akomkoľvek príspevku všetkých používateľov, a tým tiež docieliť emailovú notifikáciu o dôležitej správe. Prístup k našim konverzáciám má aj pedagogický vedúci tímu.

E-mail
Mail slúži hlavne na hromadnú komunikáciu, spolu so zadávateľom projektu. Využívali sme ho minimálne, hlavne pri zdieľaní najdôležitejších informácií. Za týmto účelom sme zriadili email:
tp07-1415@googlegroups.com

Telefón
Telefón používame len v najnutnejších prípadoch, ak člen tímu neodpodpovedá na ostatné formy komunikácie. Každý člen tímu musí povinne poskytnúť tento údaj do zoznamu, ktorý je zdieľaný na Dropboxe.

2.8 Nástroje pre zdieľanie obsahu

Google Drive
Google Drive používame pri kolaboratívnej tvorbe dokumentácie. Každý člen tímu je povinný uviesť v príslušnom súbore emailovú adresu svojho aktuálneho Google účtu.

Dropbox
Všetky súbory, ktoré súvisia s projektom, a je potrebné aby k nim mal prístup každý člen tímu, sú zdieľané prostredníctvom služby Dropbox. Každú člen tímu je povinný mať túto službu nainštalovanú a využívať ju.

3 Manažment rozsahu projektu

Najpodstatnejšou prerekvizitou správneho stanovenia rozsahu projektu je dôkladná analýza požiadaviek na výslednú funkcionalitu. Následná konštruktívna diskusia so zadávateľom projektu nás viedla k stanoveniu cieľov a otvorila nám možnosti pre analýzu dostupných riešení, z ktorej pri dodržaní nami vytvorených metodík vyplynuli postupy pre dosiahnutie týchto cieľov. Dekompozíciou takto získaných postupov sme vytvorili jednotlivé úlohy a prerozdelili sme ich medzi členov tímu. Neustálym monitorovaním a vyhodnocovaním miery splnenia úloh sme opätovne sledovali, či sme si správne stanovili rozsah našej práce. V tejto fáze, ale aj vo fázach prislúchajúcich k iným metódam riadenia projektu sme tento postup aplikovali a kontrolovali vďaka systému JIRA.

4 Manažment rozvrhu projektu

Manažment rozvrhu v našom tíme je rozdelený na niekoľko oblastí. V prvom rade je to vytváranie krátkodobého plánu na najbližší šprint, ktoré sa deje počas scrum stretnutia tímu. Na vzniku tohto plánu sa podieľa celý tím, pričom výsledná podoba úloh je vytváraná s prihliadnutím na stanovený manažment rizík. Dôležitou súčasťou je taktiež odhad časovej dotácie pre jednotlivé úlohy, pričom tieto časy odhaduje celý tím, nielen riešiteľ úlohy.

Po stretnutí je potrebné správne vytvoriť všetky identifikované úlohy v nástroji JIRA, ktorý je používaný na riadenie tímového projektu. Metodika zaoberajúca sa prácou v nástroji JIRA je popísaná v nasledujúcej kapitole. Dôležité je snažiť sa o zodpovedný prístup k práci v tomto nástroji a pravidelnú aktualizáciu stavu svojej činnosti.

4.1 Metodika vytvárania úloh v systéme JIRA

Dedikácia metodiky

Tento dokument je primárne určený pre všetkých členov tímu č.7 – BOINCTRIBUTORS, podieľajúcich sa na vývoji akýmkoľvek spôsobom. Metodika, v rámci predmetu tímový projekt, slúži na záväzné stanovenie pravidiel pri vytváraní úloh v systéme JIRA. Systém JIRA bol tímom určený ako základný nástroj na riadenie tímových činností a sledovanie tímovej aktivity.

Vytváranie úloh

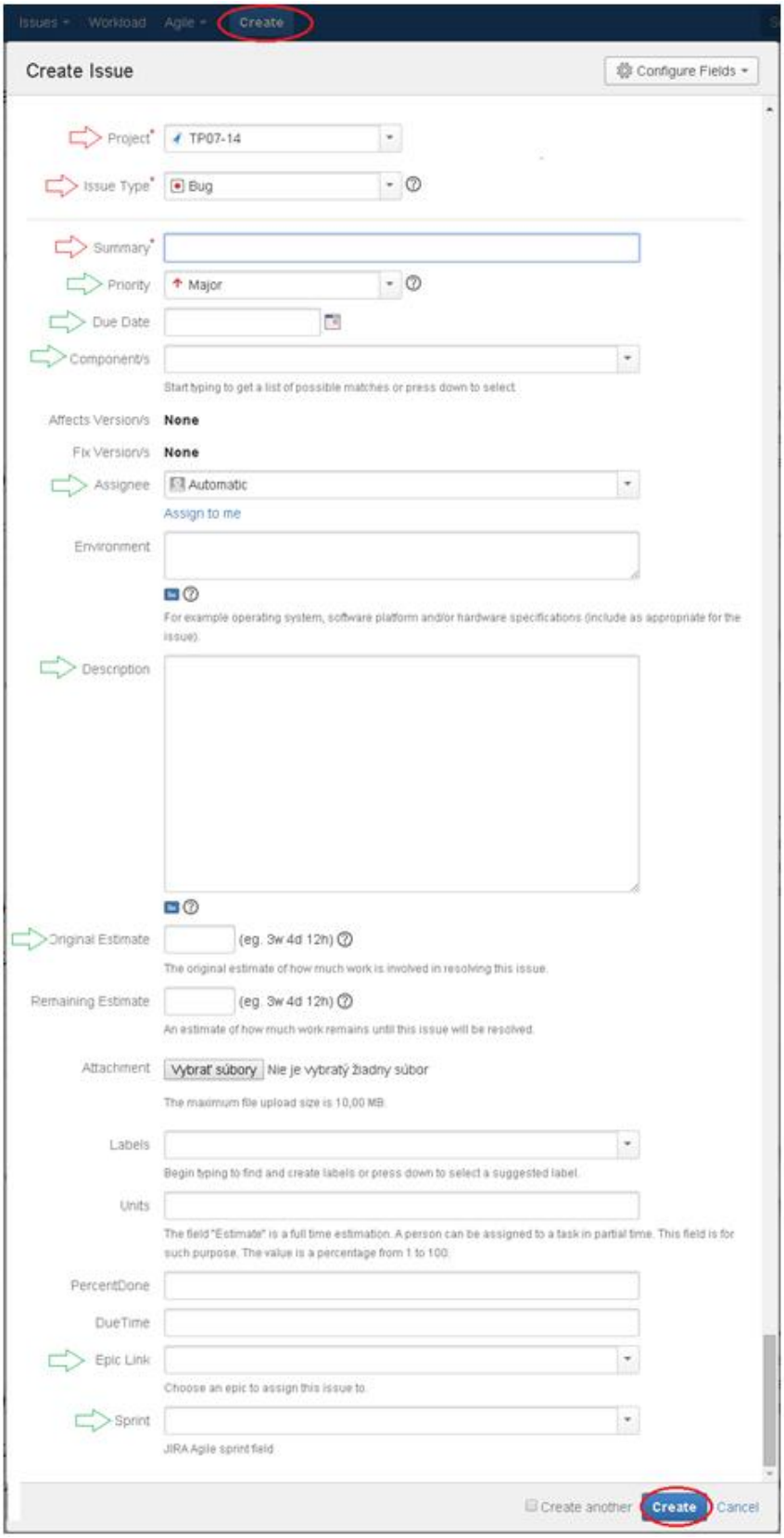
Pri vytváraní novej úlohy v systéme JIRA sa riaďte nasledovnými pokynmi a zásadami:

Po prihlásení sa do systému JIRA, v strede hornej časti stránky kliknite na “Create“, alebo použite klávesovú skratku – C.
Spustí sa dialóg vytvorenia novej úlohy.
Obrázok dialógu, so zvýraznenými relevantnými časťami, sa nachádza za touto kapitolou.
Vyplňte ho podľa nasledujúcih pokynov.
Povinné vyplniť je:
Project
Projekt, do ktorého úloha patrí.
Tento projekt je -  TP07-14
Issue type
Typ danej úlohy.
Existuje niekoľko typov úloh:
 Bug
Problém, ktorý narúša správne fungovanie projektu.
 New Feature
Problém, ktorý narúša správne fungovanie projektu.

 Task
Zadanie, ktoré je potrebné vypracovať.
Prirovnateľné kategórii “rôzne“.
 Improvement
Vylepšenie existujúcej funkcionality.
 Epic
Ucelená, veľká, samostatná funkcionalita.
Napríklad - Zaregistrovanie používateľa.
Spravidla sa následne delí na menšie úlohy, ktoré sa pod ňu zaradia.
 Story
Funkcionalita, popísaná z pohľadu koncového používateľa.
Priraďte teda úlohe najviac vyhovujúci typ.
Summary – Zhrnutie
Krátky, pár slovný opis úlohy.
Malo by byť z neho jasné kde úlohu zaradiť, nepatria tu však špecifikácie danej úlohy.
V prípade úlohy typu Epic je potrebné zadať aj názov.
Spravidla tiež vyplňte:
Priority
Priorita - dôležitosť - danej úlohy.
Stupne dôležitosti sú takéto:
 Blocker
Najvyššia priorita, takáto úloha je pre projekt kľučová a má prednosť pred všetkými ostatnými.
 Critical
Úloha predstavuje vážny problém a vyžaduje si vysokú pozornosť.
 Major
Úloha má značnú doležitosť.
 Minor
Takáto úloha má pomerne malý dopad na projekt.
 Trivial
Najnižšia priorita, jej dopad na projekt je triviálny.
Je dôležité túto kategóriu vyplniť, aby sa tím vedel správne venovať a dávať prednosť čo najdôležitejším úlohám.
Due Date
Dátum dokedy má byť úloha dokončená.
Vyplňte len ak je termín dokončenia určený.
Component/s
V tomto projekte používané na označenie kategórie, pod ktorú úloha patrí.
Napríklad – Backend, Client, Prezentácia, atď.
Assignee
Riešiteľ danej úlohy.
Vyplňte len ak je úloha určená konkrétnej osobe.

Description
Detailný popis úlohy.
Je dôležité ho vyplniť tak, aby riešiteľovi bolo jasné aké sú špecifikácie danej úlohy a čo sa od neho očakáva.
V popise je možné označiť člena tímu pomocou znaku @. Daný člen tímu bude následne o úlohe a jej stave informovaný mailom.
Original Estimate
Predpokladaný čas potrebný na dokončenie danej úlohy.
Ak nie je vyplnené zadávateľom, je nutné, aby toto pole vyplnil samotný riešiteľ úlohy.
Epic Link
Epic, pod ktorý úloh patrí.
V prípade vytvárania úlohy typu Epic, toto pole nie je.
Sprint
Šprint, do ktoré úloha patrí.
Nie všetky úlohy musia byť hneď pri vytváraní zaradené do šprintu, hlavne ak sa jedná o úlohu typu Story.
Úloha so všetkými spomínanými poľami sa považuje za správne a jednoznačne vytvorenú úlohu.
V prípade potreby je možné vyplniť aj ostatné polia, nie je to však nevyhnutné.
Úlohu je možné po vytvorení upraviť a teda prípadné chyby sú napraviteľné.

Po vyplnení všetkých špecifikácií úlohy vytvorenie potvrďte stlačením “Create“ v spodnej časti dialógu vytvorenia úlohy.



Obr. 1 Dialóg vytvorenia úlohy

Vytváranie podúloh

V sytéme JIRA je taktiež možné úlohu rozdieliť na menšie podúlohy - tzv. Sub-Task.

Sub-Task je možné vytvoriť:

Vytvorením novej podúlohy.
Cez možnosť Create Sub-Task v detaile už existujúcej úlohy.
Spustený je dialóg podobný tomu, ktorý je používaný na vytváranie klasickej úlohy. Obsahuje však menej polí na vyplnenie, najväčším rozdielom je nemožnosť zaradiť takúto podúlohu pod konkrétny Epic, keďže je automaticky zaradená pod Epic nadradenej úlohy. Nakoľko je to možné sa teda riaďte rovnakými pokynmi ako pri vytváraní úlohy.
Premenením existujúcej úlohy na podúlohu.
Cez možnosť Convert to Sub-Task v detaile už existujúcej úlohy.

Nasleduje niekoľko dialógových okien, kde je potrebné priradiť rodičovskú – nadradenú – úlohu, a kde je možné zmeniť detaily úlohy podobne ako je tomu pri vytváraní úlohy.

5 Manažment kvality

S prihliadnutím na povahu projektu sme sa zameriavali hlavne na sledovanie kvality kódu a prislúchajúcich výstupov. Najzákladnejšou metrikou merania kvality softvérových výstupov je testovanie jednotlivých funkcionalít. Na spúšťanie a manažment automatizácie testovacieho procesu sme sa rozhodli použiť nástroj Codeship, ktorý sme nakonfigurovali pre každú z vetiev spoločných repozitárov zdrojového kódu. Codeship sleduje ako internú vývojársku verziu projektových modulov, tak aj výslednú produkčnú a automatizovane spúšťa k nim prislúchajúce testovacie scenáre. Pre zlepšenie transparentnosti tohto procesu je Codeship integrovaný priamo do spoločného komunikačného nástroja tímu - Hipchat. Súčasťou tejto konfigurácie je aj automatické nasadenie produkčnej vetvy na nám pridelený virtuálny sever.

5.1 Metodika code review použitím nástroja Bitbucket

Dedikácia metodiky

Táto metodika je určená pre všetkých členov tímu, ktorí sa podieľajú na vývoji aplikácie a zahŕňa v sebe pravidlá, ako a kedy robiť *code review*. *Code review* je zjednodušene povedané pripomienkovanie kódu inými členmi tímu tak, aby kód získaval na kvalite a bol zrozumiteľný a pochopiteľný pre všetkých členov tímu.

Keďže repozitáre sa nachádzajú na bitbukete, zvolil som rozhranie tohto nástroja aj pre *code review*. Je prehľadné a úplne postačuje vzhľadom na veľkosť nášho projektu. Na nasledujúcich stranách sa nachádza postup aj s ukážkami ako robiť správne *code review*.

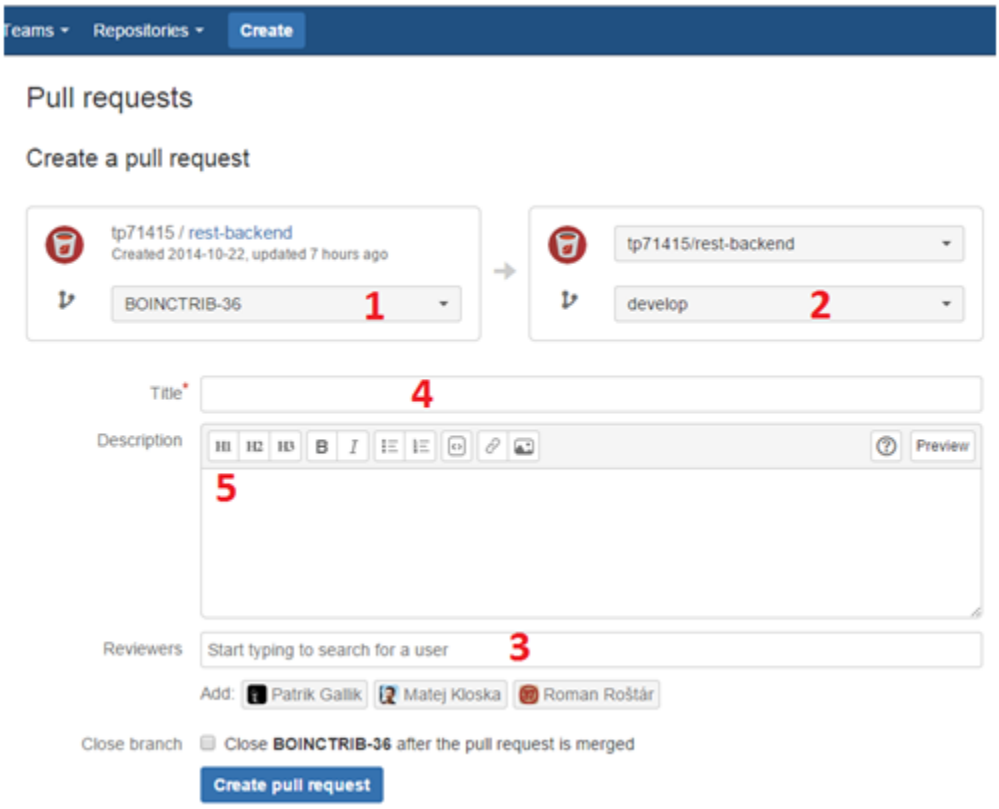
Kedy je nutné robiť *code review*

Code review je najlepšie robiť po menších celkoch, preto je potrebné tento proces vykonať vždy pri dokončení nejakej úlohy, ktorá bola zadaná v systéme Jira. Keď niektorý člen tímu začne pracovať na úlohe, vytvorí si novú vetvu z repozitára z vetvy *develop*. Následne po vyriešení tejto úlohy je povinný vykonať *pull-request*, do ktorého je povinný zadať mená minimálne dvoch členov tímu, ktorí budú hodnotiť kód napísaný pre vyriešenie tejto úlohy.

Ako vytvoriť *pull-request*

Po každom ukončení práce na úlohe je potrebné zvoliť v rozhraní bitbucket v ľavom menu voľbu create *pull-request*. Okno, ktoré sa zobrazí je na obr. 1, kde sú červenými číslami vyznačené všetky položky, ktoré je potrebné vyplniť. V rozbaľovacom zozname číslo jeden treba zvoliť vetvu, na ktorej člen tímu práve pracoval. Vpravo v rozbaľovacom zozname číslo dva treba zvoliť vetvu *develop*, prípadne inú, ak ide o iné spájanie vetiev. Pod číslom tri je označená kolónka, ktorá slúži na zvolenie minimálne dvoch členov tímu, ktorí budú kód hodnotiť. Pod číslami štyri a päť sú označené kolónky, kde sa zadáva nadpis a popis, pre spojenie vetiev. Tu je riešiteľ povinný popísať, čo sa v danej úlohe riešilo, prípadne ak ide o zložitejšie riešenie načrtnúť stručnú myšlienku riešenia, aby bol ostatným členom tímu, pri hodnotení

kódu, zrejmý myšlienkový postup riešenia časti úlohy. Po vyplnení formuláru riešiteľ potvrdí *pull-request* a členovia tímu sú upozornení mailom o tom, že bol *pull-request* otvorený.

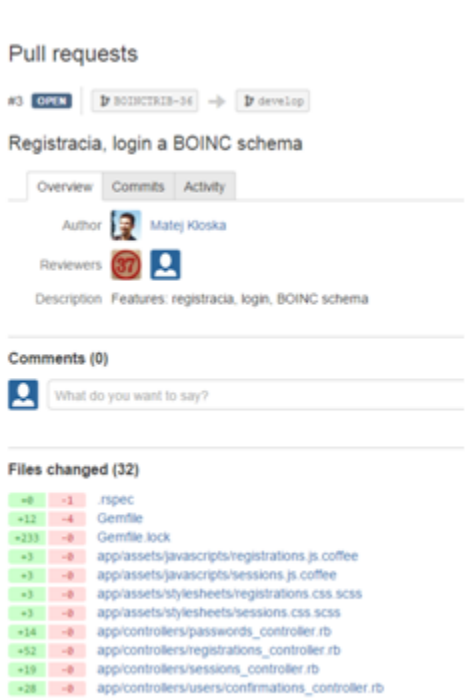


Obr. 1 Formulár pre pull request.

Práca s pull-request

Členovia tímu si po prijatí mailu nájdu *pull-request* v systéme bitbucket. V ľavom menu vyberú voľbu *pull-request* a následne sa im zobrazí zoznam, kde si vyberú daný *pull-request*, ako je možné vidieť na obr. 2. Daný *pull-request* sa otvorí, tak ako je možné vidieť na obr. 3. V základnom pohľade je možné vidieť zoznam zmenených súborov a pod nimi sú zmeny v daných súboroch rozpísané detailne, ako zobrazuje obr. 4.

Obr. 2.: Pridávanie komentárov



Obr. 3.: Prehľad pull requestov

Obr. 4.: Detaily vybraného pull requestu

Komentáre je možné pridávať na dvoch miestach, a síce k celému balíku zmien, ktorý obsahuje *pull-request* a to tak, že komentár pridáme hore nad zmenené súbory ako je znázornené na obr. 2. Na tomto mieste môžeme vidieť všetky už pridané komentáre k akcii *pull-request*.

Ďalším spôsobom je pridanie komentáru ku konkrétnemu riadku kódu. Pod zoznamom zmenených súborov, ktorý je vidieť na obr. 3 dole, je každý zmenený súbor rozpísaný osobitne. Na obr. 4 je možné vidieť súbor, ktorý bol upravovaný. Pokiaľ chce člen tímu pridať komentár ku konkrétnemu riadku kódu, klikne myšou na ikonu plus, ktorá sa nachádza vľavo v príslušnom riadku kódu, viď obr. 4 vľavo.

Po kliknutí myšou na ikonu sa na danom riadku otvorí dialógové okno, do ktorého člen tímu napíše komentár k danému riadku, viď obr. 5. Pre komentovanie kódu je možné využiť základné značky, ktoré poskytuje rozhranie nástroja bitbucket, taktiež vložiť nejaký odkaz alebo aj obrázok. Po napísaní komentáru ho stačí potvrdiť modrým tlačidlom *Comment*.

Tak ako je možné pridávanie komentárov k riadkom a celej akcii *pull-request*, je možné komentovanie aj jednotlivých commit akcií, ktoré sú súčasťou daného celku. Na obr. 3 je možné vidieť, že doteraz sa

pracovalo na karte *Overview*. Po prepnutí na kartu *Commits* sa zobrazia všetky *commit* akcie, ktoré boli vykonané, ako znázorňuje obr. 6. Po kliknutí na daný *commit* sa rozbalia všetky súbory, ktoré boli zmenené v danej *commit* akcii a je ich možné komentovať vyššie uvedeným spôsobom.

Komentáre sa pridávajú k daným riadok vtedy, pokiaľ daný riadok nespĺňa vopred dohodnuté štandardy. Ide o všeobecne známe štandardy daných jazykov, v ktorých je kód napísaný. Pokiaľ je daný kód napísaný v jazyku ruby, tak sa komentujú riadky, ktoré porušujú tento štandard. Pokiaľ ide o súbory, v ktorých je písaný sass, je potrebné sa riadiť metodikou k písaniu kaskádových štýlov v sass.

Ukončenie *code review*

Ak člen tímu skontroloval kódy a okomentoval ich, má na výber niekoľko možností. Pokiaľ sa v kóde nachádzali chyby, ktoré bránia tomu, aby bola daná vetva spojená s vyššou vetvou (napr. *devel*), zvolí voľbu *Decline*, ktorá je umiestnená v pravom hornom rohu, viď obr. 6. V prípade, že s kódom súhlasí, zvolí voľbu *Approve*. Ak kód hodnotí a komentuje posledný člen tímu a zhodnotí ho ako dobrý a všetci členovia pred ním s kódom súhlasili, zvolí voľbu *Merge*, čím potvrdí spojenie danej vetvy s vyššou vetvou (napr. *devel*).

6 Manažment rizík

Dôležitou časťou manažmentu tímu v tímovom projekte je aj manažment rizík. V našom projekte sme doposiaľ určili niekoľko rizík, ktoré sa pri vývoji softvéru bežne objavujú. Takéto riziká su nazývané generickými, teda všeobecnými. Najčastejšie z nich sme, spolu so špecifickými rizikami identifikovanými vrámci nášho projektu sme spísali do nasledujúcej tabuľky, pričom našim cieľom je čo najviac znížiť pravdepodobnosť výskytu generických rizík.

	Prevencia	**Popis**	**Dopad**	**Pravdepodobnosť**	**Typ**	**Riziko**
1	Agilný vývoj, JIRA	Riziko zlého plánovania z časového hľadiska, ako aj z hľadiska výstupov realizovateľných úloh.	Stredný	Stredná	Generické	Nerealizovateľné plánovanie
2	Komunikácia, JIRA, Tímové stretnutia	Riziko, že viacerí členovia tímu budú duplicitne pracovať na jednej úlohe.	Stredný	Nízka	Generické	Nadbytočná práca
3	Dôsledná analýza, Codereview	Riziko nestabilnosti softvéru a problémov pri pridávaní nových funkcionalít.	Stredný	Stredná	Generické	Kvalita a udržateľnosť softvéru
4	Dôsledná analýza, Modulárnosť	Riziko použitia zlej alebo nevyhovujúcej architektúry v projekte.	Vysoký	Nízka	Generické	Nevhodná architektúra

5	Zapojenie potenciálnych používateľov k interakcii so systémom, Návrhy reflektujúce prípady použitia	Riziko komplikovaného alebo nedostatočne prehľadného používateľského rozhrania.	Stredný	Stredná	Generické	Neprehľadné používateľské rozhranie
6	Pozitívna morálka, Teambuilding	Riziko odchodu niektorého z členov tímu.	Vysoký	Nízka	Generické	Odchod členov tímu
7	Dôsledné záťažové testovanie	Riziko možných výpadkov servera alebo výpadkov v dôsledku interných chýb aplikácie.	Vysoký	Stredná	Generické	Nedostupnosť kvôli servrovým a interným chybám
8	Pravidelná komunikácia s vedúcim tímu, Dôsledné zhodnocovanie stavu projektu	Riziko neporozumenia a nedodržania požiadaviek na produkt, ktoré určil jeho vlastník.	Stredný	Nízka	Generické	Nesplnenie požiadaviek vlastníka produktu
9	Propagácia projektu a osveta v rámci fakulty, Motivujúci informačný obsah domovskej stránky projektu	Riziko malého záujmu používateľov o zapojenie sa do projektu formou dobrovoľného poskytnutia výpočtového výkonu svojho počítača.	Vysoký	Vysoká	Špecifické	Nedostatočný počet dobrovoľníkov participujúcich na fakultnom projekte BOINC@FIIT

7 Manažment integrácie projektu

Na zaistenie správnej koordinácie súbežných procesov vyskytujúcich sa počas riadenia tímového projektu a na zaistenie požadovanej kolaborácie a kooperácie jednotlivých modulov tvoriacich systém sme stanovili nasledovný postup: Každému z logicky oddeliteľných procesov, či už administratívnych alebo implementačných sme pridelili jedného člena tímu, ktorý je zodpovedný za kvalitatívny výstup daného procesu. Jednotliví vedúci takto určených elementov projektu spolu intenzívne komunikujú a zabezpečujú tak potrebnú kolaboráciu. Vedúci tímu, po dohode s mentormi procesov, stanoví výslednú podobu procesu.

7.1 Metodika práce s migráciami v prostredí webového rámca Ruby on Rails

Úvod

1.1 Dedikácia metodiky

Táto metodika obsahuje popis postupov pri práci s dátovým modelom, ktoré je nutné počas vývoja dodržiavať. Primárne je určená pre členov tímu TP 07, no vďaka použitiu všeobecných pravidiel je možné dokument aplikovať taktiež pri vývoji iných aplikácií používajúcich technológie použité pri opise. Čitateľ po preštudovaní získa informácie o najčastejších procesoch súvisiacich s úpravou dátového modelu aplikácie spolu s modifikáciou dát v určitých scenároch, keď je to nevyhnutné. Nástroj pre prácu s migráciami je štandardný migračný subsystém obsiahnutý vo webovom rámci Ruby on Rails. Celá metodika sa vzťahuje na spomínaný migračný subsystém a nie je možné ju uplatniť na iný ekvivalentný nástroj[1].

Dokument nepokrýva základné princípy práce s webovým rámcom Ruby on Rails a nástrojom pre správu verzií zdrojového kódu - Git.

1.2 Použité pojmy

rake (ruby make): štandardný Ruby nástroj, ktorý nahrádza známy UNIX nástroj s názvom *make* a rozširuje jeho funkčnosť. V prípade Ruby On Rails aplikácií poskytuje rozhranie pre spúšťanie opakovaných úloh, s ktorými sa vývojár stretáva počas celého vývojového cyklu,
git: distribuovaný systém riadenia revízií navrhnutý pre správu a riadenie revízií zmien malých, stredných i veľkých projektu,
commit: zapísanie zmien v kóde do systému git,
rails (ruby on rails): rámec pre vývoj webových aplikácií pomocou programovacieho jazyka Ruby
migračný súbor: ruby skript, ktorý obsahuje postupnosť migračných krokov, ktoré je potrebné vykonať pre dosiahnutie požadovanej zmeny na úrovni databázy,

[1] V čase zostavovania tejto metodiky neexistoval alternatívny nástroj pre spravovanie databázových migrácií pre webový rámec Ruby on Rails.

vzdialený repozitár - vzdialený priestor (adresár) na serveri, kde git ukladá zdrojové kódy, pomocné súbory projektu a metadata o verziách.
lokálny repozitár - vzdialený priestor (adresár) na serveri, kde git ukladá zdrojové kódy, pomocné súbory projektu a metadata o verziách.

Všeobecný popis migrácií a pravidiel

Každý zadávaný text či už do konzoly alebo zdrojového súboru migrácie zapíšte v anglickom jazyku. Pokiaľ je možné, použite základné známe slovesá popisujúce akciu alebo akcie vykonávajúce sa v danej migrácií ako napríklad:

pridaj - *add*,
vytvor - *create*,
premenuj - *rename*,
odstráň - *remove*.

V prípade použitia vyššie uvedených akcií v správnom tvare, migračný systém dokáže odvodiť z názvu akciu a spätnú akciu, ktoré je potrebné vykonať.

V prípade migrácií je dôležité uvedomiť si časový rozmer každej migrácie. Pri vytvorení migračného súboru je k definícií migrácie priradená časová pečiatka popisujúca dátum a čas, kedy bolo daný súbor vytvorený. Pri aplikovaní migrácií sú dostupné migrácie zoradené vzhľadom na pečiatku od najstarších po najnovšie. V tomto poradí sú následne aplikované na zadanú databázovú schému.

2.1 Pomenovanie tabuliek

Pre vhodné odvodenie názvu tabuľky vychádzame z pomenovania entity, ktorú chceme reprezentovať danou tabuľkou postupujte podľa nasledujúcich krokov:

anglický názov tabuľky prevedťe do plurálu, ak sa tak nestalo už v základnom stave,
názov tabuľky odvoďte spojením všetkých slov malými písmenami pomocou "_".

V prípade pomenovania väzobnej tabuľky postupujte samostatným pomenovaním oboch tabuliek podľa vyššie spomenutých pravidiel a následného spojenia názvov pomocou "_".

```
Entita: Vlastnosť produktu
Anglický ekvivalent: Product Property
Množné číslo: Product Properties
Názov tabuľky: product_properties
```

2.2 Pomenovanie stĺpcov

Pre správne pomenovanie stĺpca vezmite za základ, s ktorým budeme ďalej pracovať názov vlastnosti entity. Anglický názov vlastnosti tak ako aj v prípade tabuliek spojte pomocou "_" a samozrejme preveďte na malé písmená. Výnimky pre stĺpce so špeciálnym významom:
primárny kľúč: vždy použite pomenovanie *id*. Ak je naozaj nutné porušiť toto pravidlo, je nevyhnutné uviesť patričnú poznámku pri danom stĺpci v databázovej tabuľke, migrácii a samotnom modeli,
časový stĺpec: pri vyjadrení určitého presného času, ktorý je dôležitý z pohľadu daného záznamu použite pri pomenovaní príponu "*_at*"

```
Vlastnosť: Výtvorený
Anglický ekvivalent: Created
Stĺpec: created_at
```

referencia na inú tabuľku (cudzí kľúč): pri vyjadrení referencie na inú tabuľku postupujte zložením názvu z jednotného čísla referencovanej tabuľky s príponou "*_id*"

```
Referencia v tabuľke profiles na tabuľku users:
Stĺpec: user_id
```

```
class ExampleMigration < ActiveRecord::Migration
  def up
    create_table :profiles do |t|
      t.facebook :string
      t.twitter :string
      t.facebook :string
    end

    add_column :users, :locked, :boolean
    rename_column :users, :name, :first_name
  end

  def down
    rename_column :users, :first_name, :name
    remove_column :users, :locked

    drop_table :profiles
  end
end
```

Príklad kompletnej migrácie, ktorá vytvára tabuľku, pridáva a premenováva stĺpec.

Postupy

3.1 Vytvorenie novej migrácie[2]

Vstup: požiadavka na vytvorenie migračného skriptu pre možnosť opätovného aplikovania zmeny na databázu
Výstup: migračný súbor pripravený pre ďalšiu prípadnú úpravu a následné aplikovanie

Ak vznikne počas vývoja potreba vykonať zmenu v databázovej schéme, pokračujte podľa nasledujúcich krokov pre vytvorenie korektnej znovu použiteľnej migrácie. Základný postup pozostávajúci z niekoľkých krokov popíšeme na príklade pridania stĺpca reprezentujúceho jednoduché počítadlo prihlásení do tabuľky users:

vytvorte migračný súbor pomocou príkazu:

```
$ rails generate migration AddSignInCountToUsers
invoke  active_record
create    db/migrate/20141116164207_add_sign_in_count_to_users.rb
```

otvorte súbor 20141116164207_add_sign_in_count_to_users.rb, ktorý bude obsahovať prázdnu migráciu podobnú ukážke nižšie

```
class AddSignInCountToUsers < ActiveRecord::Migration
 def change
 end
end
```

do metódy change zadajte požadovanú zmenu (v našom prípade pridanie stĺpca):

```
class AddSignInCountToUsers < ActiveRecord::Migration
 def change
   add_column :users, :sign_in_count, :integer
 end
end
```

uplatnite danú migráciu pomocou príkazu (pre viac informácií, pokračujte podkapitolou 4.2 Uplatnenie migrácií v tomto dokumente):

$ rake db:migrate

[2] Oficiálna dokumentácia k vytváraniu migrácií:
http://edgeguides.rubyonrails.org/active_record_migrations.html#creating-a-migration

3.2 Uplatnenie migrácií

Vstup: prevzatie zmien zo vzdialeneho repozitára
Výstup: aktuálna databázová schema vzhľadom na dostupné migrácie

Pre uplatnenie migrácií od iný vývojárov v tíme je postačujúce zadať príkaz:

```
$ rake db:migrate
```

3.3 Úprava existujúcej migrácie

Vstup: identifikovanie chyby v existujúcej migrácií
Výstup: úpravená existujúca migrácia alebo nová migrácia

Počas vývoja môže nastať situácia, kedy bude potrebné vykonať zmenu v už existujúcej migrácii. Tento postup je možný len a len vtedy, keď nebol vykonaný commit a následný push do vzdialeného repozitára. V prípade, že ste identifikovali problém u seba lokálne postupujte nasledovne:

zrušte migráciu v prípade, že ste ju aplikovali na schému pomocou príkazu
upravte migráciu
opätovne aplikujte migráciu
vykonajte commit s daným migračným súborom

V prípade, že ste vykonali commit s migračným súborom, v ktorom ste identifikovali chybu, vykonajte čo najskôr nápravu. Vytvorením novej migrácie pomocou postupu zadefinovaného v 4.1 Vytvorenie novej migrácie. Na začiaku migračného súboru je nutné uviesť komentár obsahujúci poznámku popisujúcu, že sa jedná o opravnú migráciu spolu s názvom migrácie, ktorú opravuje.

```
# FIX FOR 20141116164207_add_sign_in_count_to_users
class ChangeSignInCountInUsers < ActiveRecord::Migration
 def change
 end
end
```

3.4 Odstránenie migrácie

Vstup: potreba odstrániť migráciu resp. tabuľku / stĺpec z databázy
Výstup: odstránené objekty alebo žiadna zmena

Odstránenie migrácie možete vykonať len v prípade, že nebol vykonaný commit s daným migračným súborom. Ak áno, jedinou možnosťou je vytvoriť migráciu, ktorá vykoná odstránenie daných databázových prvkov. Určite nie zmazať fyzicky danú migráciu, nakoľko týmto krokom môžete vniesť značnú nekonzistenciu do databázových schém na rôznych miestach.

V prípade, že ste vykonali zmenu schémy len u seba lokálne, postupujte nasledovne:

zrušte zmenu v schéme pomocou príkazu:

```
$ rake db:rollback
```

fyzicky zmažte súbor
spustite príkaz pre opätovné zmigrovanie databázovej schémy:

```
$ rake db:migrate
```

v prípade, že ste vykonali commit migračného súbor do lokálneho repozitára, vykonajte opravný commit s odstránením súboru.

8 Manažment softvéru

Pre sprehľadnenie procesu kolaboratívnej tvorby softvérového produktu sa štandardne používa niektorý z dostupných nástrojov na verziovanie softvéru. My sme sa rozhodli použiť nástroj Bitbucket na vytváranie a správu našich repozitárov zdrojového kódu. Poskytuje jednoduché rozhranie pre vytváranie pull requestov a zároveň poskytuje aj možnosť prehliadky kódu. Vďaka týmto funkciám nám tento nástroj umožňuje dodržiavať našu interne dohodnutú metodiku verziovania zdrojového kódu, pričom okrem spomínaných benefitov poskytuje pri akýchkoľvek zmenách vo vetvách repozitárov priamu integráciu v tímovom komunikačnom nástroji - Hipchat.

8.1 Metodika vetvenia a udržiavania zdrojového kódu

Úvod

Preambula

Táto metodika spadá pod manažment podpory vývoja a je určená pre všetkých členov tímu BOINCTRIBUTORS, ako aj pre všetkých iných prispievateľov do tohto projektu. Cieľom tejto metodiky, je stanovenie pravidiel pre udržiavanie konzistentnej bázy kódu v rámci systému pre verziovanie kódu. V našom prípade používame ako VCS program git s centrálnym úložiskom v službe bitbucket.org.

Pre rozumné rozvrhnutie manažmentu vetvenia kódu sme sa inšpirovali už existujúcou a overenou metodikou Vincenta Driessena[3], ktorú sme si upravili pre naše potreby, ako aj pokynmi a odporučeniami spomínanými v oficiálnej dokumentácii služby bitbucket.org.

Použité skratky

VCS (version control system) – systém pre udržiavanie verzií
PMS (project management system) – systém pre manažment projektu
git – VCS program
branch – vetva kódu
commit – odovzdanie zmien v kóde

Prerekvizity

Pre potreby tejto metodiky sa predpokladá, že čitateľ má na svojom počítači nainštalovaný program git, so všetkými závislosťami. Podrobné inštalačné inštrukcie nájdete na oficiálnej stránke programu git.

Taktiež predpokladáme základnú schopnosť práce s gitom, na úrovni try.github.io.

[3] http://nvie.com/posts/a-successful-git-branching-model/

Vetvenie kódu

Základné pravidlá pre vetvenie kódu

Pre udržiavanie konzistentnej bázy kódu je v prvom rade dôležite dodržiavať pravidlá a odporúčania pre rozumné pomenúvanie vetiev kódu. Vetvy kódu musia byť vždy keď je to možné, prepojené na konkrétnu úlohu v PMS, v našom prípade ako PMS používame softvérAtlassian JIRA[4]. Pri pomenovávaní vetvy sa riaďte nasledovnými pravidlami:

názov úlohy sa musí skladať z dvoch častí [typ_ulohy]/[id_v_pms]
typy úloh môžu byť
task
feature
hotfix
story
typ úlohy musí byť napísaný malými písmenami
v prípade feature vetiev je prípustné vynechať pomenovanie typu úlohy

Napríklad, ak sa v rámci vetvy rieši nová funkcionalita prihlásenie používateľav rámci karty s id 47 (v systéme JIRA označená ako MENOPROJEKTU-47), tak príslušná vetva by sa mala nazvať:

```
feature/MENOPROJEKTU-47
```

alebo

```
MENOPROJEKTU-47
```

Nekorektné pomenovanie by v tomto prípade bolo:

```
~~Prihlásenie používateľa~~
```

Základné typy vetiev kódu

Pri vývoji projektu sa udržujú 4 základné typy vetiev kódu a to:

master (hlavná vetva)
hlavná vetva kódu, tato vetva je nasadená v ostrej prevádzke a je viditeľná pre zákazníka
táto vetva musí byť vždy nasaditeľná a musí byť maximálne funkčná
· development (vývojová vetva)

[4] https://www.atlassian.com/software/jira

vývojová vetva odzrkadľuje aktuálny stav projektu
tato vetva je nasadená v testovacom prostredí (angl. staging)
po otestovaní v testovacom prostredí sa spája do hlavnej vetvy
feature branches (vetvy s novou funkcionalitou)
tieto vetvy vznikajú z vývojovej vetvy
v jednej vetve sa implementuje jedna konkrétna funkcionalita
spájajú sa do vývojovej vetvy
hotfixbranches
tieto vetvy vznikajú pre potreby kriticky dôležitých opráv hlavnej (master) vetvy
spájajú sa prednostne do hlavnej vetvy a tiež do vývojovej vetvy

Pravidlá pre odovzdávanie

Odovzdania (angl. commits) by sa mali vždy týkať jednej konkrétnej zmeny. Pri jednotlivých odovzdaniach sa môžu meniť viaceré zdrojové súbory, ale tieto zmeny musia spolu súvisieť. Pri odovzdávaní a písaní odovzdávacej správy (commitmessage) vždy podľa týchto pravidiel:

odovzdávacie správy sa musia písať vždy v anglickom jazyku
v odovzdávaných zmenách sa nemôže nachádzať zakomentovaný kód
pri odovzdávaní do feature a hotfix vetiev na koniec odovzdávacej správy vždy pridájte „—skip-ci“ pre preskočenie kontinuálnej integrácie nástrojom codeship.io[5]
správa by mala stručne ale vecne opisovať zmeny vykonané v kóde

Práca na novej funkcionalite (features)

Pri práci na novej funkcionalite musíte najprv vytvoriť novú feature vetvu z vývojovej vetvy

```
>git checkout –b feature/BOINCTRIB-1 develop
```

Prácu priebežne odovzdávajte

```
>git add --all
>git commit –m „messageaboutprogress on the feature --skip-ci“
```

Odovzdávajte aj do príslušnej vetvy v centrálnom repozitári.

```
>git pushorigin feature/BOINCTRIB-1
```

[5] https://codeship.com/

Aby ste pracovali s čo najaktálnejšou verziou vývojovej vetvy, robte pravidelne rebase vašej feature vetvy s vývojovou vetvou.

```
>git checkout develop
>git pull
>*...vyriešte prípadné konflikty*
>git checkout feature/BOINCTRIB-1
>git rebase develop
```

Po ukončení práce na novej funkcionalite vytvorte pull request podľa pravidiel príslušnej metodiky. Po revízii kódu a akceptovaní zmien vykonaných v feature vetve zlúčte feature vetvu a vývojovú vetvu.

```
>git checkout develop
>git merge feature/BOINCTRIB-1
```

Ak nastanú konflikty pri zlučovaní vetiev, vyriešte konflikty a následne publikujte vývojovú vetvu.

```
>git push origin develop
```

Oprava kritickej časti kódu (hotfixes)

Aj keď sa vývoj aplikácie riadi podľa stanovených pravidiel, môže nastať situácia, že sa do produkčnej (hlavnej) vetvy dostane kód, ktorý aj napriek tomu, že prešiel testovaním spôsobí chyby. V takom prípade sa takáto závažná chyba rieši pomocou hotfix vetvy.

Hotfix vetvu vytvorte z hlavnej vetvy.

```
>git checkout –b hotfix/BOINCTRIB-2 master
```

Priebežne odovzdávajte vami vykonané zmeny.

```
>git add --all
>git commit –m „fixes severe productionbugs --skip-ci“
```

Vaše odovzdania priebežne pushujte do zodpovedajúcej vetvy centrálneho úložiska.

```
>git push –u originhotfix/BOINCTRIB-2
```

Následne vytvorte pull request pre zjednotenie do hlavnej vetvy, podľa pravidiel prislúchajúcej metodiky a zrevidujte ich. Po úspešnej revízii kódu zlúčte hotfix vetvu do hlavnej vetvy.

```
>git checkout master
>git merge --no-ffhotfix/BOINCTRIB-2
```

Po zlúčení s hlavnou vetvou zlúčte hotfix vetvu aj s vývojovou vetvou.

```
>git checkout develop
>git merge --no-ffhotfix/BOINCTRIB-2
```

Vyriešte prípadné konflikty a publikujte vývojovú a hlavnú vetvu.

```
>git checkout master
>git push origin master
>
>git checkout develop
>git push origin develop
```

8.2 Metodika písania kaskádových štýlov použitím nástroja SASS

Úvod

Dedikácia metodiky

Tento dokument je primárne určený pre všetkých členov tímu č.7 – BOINCTRIBUTORS, podieľajúcich sa vývoji a úpravách rozhraní hlavnej aplikácie. Metodika, v rámci predmetu tímový projekt, slúži na záväzné stanovenie pravidiel pri písaní kaskádových štýlov. Nástroj Sass bol tímom určený ako základný nástroj pre vytváranie kaskádových štýlov, najmä pre jeho jednoduchú inštaláciu v prostredí Ruby, Ruby on Rails projektov.

Prerekvizity

Na používanie Sass je nutné mať nainštalované Ruby. Inštalácia Ruby sa líši na základe platformy, na ktorú bude nainštalované. Postupujte nasledovne:

Windows: Použite *RubyInstaller*, dostupný na: *http://rubyinstaller.org/*

Linux:

Debian, Ubuntu:

```
$ sudo apt-get install ruby-full
```

CentOS, Fedora:

```
$ sudo yum install ruby
```

Mac OS X:
Predinštalovaný: 2.0 (OS X Mavericks), 1.8.7 (OS X Mountain Lion), Pre nainštalovanie najnovšej verzie použite Homebrew:

```
$ brew install ruby
```

Použitím systémovej konzoly vykonajte samotnú inštaláciu nástroja Sass (súčasť rámca Compass):

Windows:

```
gem install compass
```

Linux/Mac OS X:

```
sudo gem install compass
```

Zdrojové súbory *.scss* kompilujte do *.css* využitím príkazu:

```
compass watch
```

Zoznam nadväzujúcich metodík

Patrik Gallik – Vývoj SPA aplikácie

Vymedzenie pojmov, skratiek

DOM – Document object model – objektová reprezentácia (nielen) HTML dokumentu, zväčša v podobe stromu, ukladaná do pamäte tak, aby bolo možné jednoducho pristupovať k jednotlivým elementom dokumentu na akejkoľvek úrovni.

.scss – Prípona kaskádových štýlov vytváraných nástrojom Sass.

.css – Štandardná prípona kaskádových štýlov.

Štruktúra kaskádových štýlov

Pri vytváraní štruktúry musíte dbať na to, aby boli vaše kaskádové štýly čo najviac znovupoužiteľné, no zároveň aby neboli príliš všeobecné a aby každá deklarácia potomkov nespôsobila preťaženie týchto štýlov. Zároveň nie je dobré odkopírovanie DOM štruktúry HTML dokumentu, nakoľko by vaše štýly

neboli znovupoužiteľné vôbec, a akákoľvek zmena v HTML by mala za následok zmeny v .scss a naopak. Pri stanovení štruktúry prihliadajte na charakter aplikácie, a ako aj v našom prípade, je vhodné rozdeliť si štýly modulárne. Rozdeľte ich nasledovne:

```
stylesheets/
|
|-- admin/        # Adminská sekcia pre zadávateľa úloh
|   |-- modules/
|   |-- partials/
|   `-- _base.scss
|
|-- account/      # Používateľská sekcia pre participantov na úlohách
|   |-- modules/
|   |-- partials/
|   `-- _base.scss
|
|-- site/         # Stránka projektu
|   |-- modules/
|   |-- partials/
|   `-- _base.scss
|
|-- vendor/       # CSS alebo Sass tretích strán
|   |-- bootstrap/
|   |-- _jquery.ui.core-1.9.1.scss
|   ...
|
|-- admin.scss    # Primárne štýly pre jednotlivé sekcie
|-- account.scss
|-- site.scss
```

Používanie nosných knštrukcií nástroja Sass

Vhniezdenie

Jednou z najužitočnejších funkcionalít nástroja Sass je možnosť vhniezdňovať do seba jednotlivé štýly. Dajte si však pozor, ako aj pri deklarácii štruktúry štýlov, na odkopírovanie štruktúry HTML dokumentu, nakoľko pri spätnej kompilácii na .css by vygenerovaný dokument mohol pôsobiť neprehľadne.

Vhniezdňujte preto maximálne do 4. úrovne, a pri akejkoľvek vyššej úrovni sa snažte si tento spôsob odôvodniť, no výrazne neodporúčame vhniezďovať hlbšie. Pri hlbokom vhniezdnení vznikajú zbytočne veľké súbory definujúce štýly, čo môže pri veľkom projekte zohrať kľúčovú úlohu v rámci optimalizácie zobrazenia webových stránok aj pri pomalšom pripojení.

Premenné

Špecifické premenné deklarujte pre každý modul zvlášť, globálne premenné si udržiavajte na jednom mieste pre zachovanie modularity a konfigurovateľnosti. V prípade zmeny v štruktúre stránky bude tak potrebné túto zmenu spraviť len na jednom mieste. V ostatných súboroch .scss potom už iba použijete klauzulu @Import a premenné z ostatných modulov budú viditeľné a použiteľné. Používajte čo najviac významové, prefixové a sufixové pomenovávanie, samozrejmosťou je používanie anglických názvov premenných. Dizajnérsky, udržiavajte si napríklad hlavné farby vo významom pomenovaní, a to napr.:

```
// Zlá deklarácia              // Príklad prefix “header”
$red: red;                     $header-height: 100px;
$yellow: yellow;               $header-background-color: $brand-color;

// Lepšia deklarácia
$brand-color: red;
$accent-color: yellow;
```

Mixins

Mixin je nesmierne silný nástroj, no s veľkou vyjadrovacou silou prichádza veľká zodpovednosť. V podstate ide o deklaráciu funkcionality, a pri jej neskoršom zavolaní s deklarovanými parametrami a hodnotami sa tieto údaje dosadia podľa deklarácii. Pri kompilácii na .css súbor sa údaje dosadia ako v prípade akéhokoľvek šablónovacieho nástroja. Pri vytváraní vlastných deklarácii najprv overte, či už neexistuje rovnaká, poprípade lepšia deklarácia, a ak neexistuje až potom vytvárajte vlastnú deklaráciu. Využite nástroj Bourbon na vyhľadanie dostupných mixin deklarácií.

```
/*MIXIN deklarovaný globálne*/
@mixin css3-calc($property, $operation) {
  #{$property}: -moz-calc(#{$operation});
  #{$property}: -webkit-calc(#{$operation});
  #{$property}: calc(#{$operation});
}

/*Použitie MIXIN*/
.content {
  @include css3-calc('width','100% - 200px');
}

/*Výstup MIXIN vo formáte CSS*/
.content {
  width: -moz-calc('100% - 200px');
  width: -webkit-calc('100% - 200px');
```

```
  width: calc('100% - 200px');
}
```

Mixin zachováva základné pravidlo o modularite, kde zmena deklarácie štýlov, napr. vynechaním prefixu prehliadača pri niektorých atribútoch by mala za následok, v normálnom prípade, vyhľadanie a prepísanie množstva kódu, použitím *Mixins* by to znamenalo zmenu jedného, deklaračného, riadku. Preto používajte čo v najväčšej miere nástroje podporujúce modularitu a konfigurovateľnosť.

8.3 Vývoj a údržba klientskej časti aplikácie projektu BOINC

Úvod

Dedikácia metodiky

Tento dokument obsahuje postupy, ktoré sú povinní dodržiavať všetci členovia tímu pri upravovaní a pridávaní funkcionality do klientskej časti BOINC aplikácie (ďalej len aplikácie). Tento dokument nerozoberá spôsob správneho písania kódu, je odporúčané používať "best practises" rámcov (frameworkov), ktoré používame.

Krátko o aplikácii

Aplikácia je vyvíjaná pomocou frameworku AngularJS (https://angularjs.org/). Dokumentáciu k frameworku môžete nájst tu (https://docs.angularjs.org/). SPA je napojená na REST API, ktoré poskytuje serverová aplikácia v repozitári rest-backend. Preto je pri vývoji potrebné mať túto serverovú aplikácie lokálne spustenú, viac informácií v readme súbore spomenutého repozitára.

Štruktúra aplikácie

V projekte musí byť dodržaná nasledujúca adresárová štruktúra:

spa-client
app - obsahuje všetky zdrojové kódy aplikácie
images - statické obrázky
scripts - javascript súbory
controllers - ovládače frameworku Angular
services - služby framework Angular
styles - css súbory
views - html šablóny
bower_components - obsahuje externé knižnice
dist - obsahuje zbuildovanú aplikáciu
node_module - obsahuje nodejs moduly

test - obsahuje testovacie súbory
GruntFile.js - obsahuje zadefinové Grunt úlohy, ktoré sa starajú o vytvorenie servera, build, test aplikácie (viď nižšie)
bower.json - obsahuje zoznam modulov tretích strán, ktoré spravuje nástroj Bower
package.json - zoznam závislosti pre NPM (NodeJS moduly), ako napríklad grunt, grunt úlohy, bower

Priečinky označené kurzívou sa nenachádzajú v repozitári projektu, každý vývojár si ich vytvára lokálne, pomocou nástrojov opísaných nižšie. Vyššie uvedená štruktúra obsahuje aj opis niektorých iných dôležitých súborov, potrebných pre vývoj.

Inicializácia a nastavenie projektu

Na vyvíjanie projektu je potebné mať nainštalovaný NodeJS (http://nodejs.org/) a jeho package manager NPM. V prípade, že nemáte nainštalovaný NodeJS a NPM, najjednoduchšie je si stiahnuť inštalačný balíček (http://nodejs.org/download/) a postupovať podľa pokynov.

Medzi NodeJS moduly, ktoré je potrebné mať nainštalované (globálne), patria:

- Grunt - nástroj na automatizáciu úloh (http://gruntjs.com/)
- Bower - správa front-end pluginov tretích strán (http://bower.io/)
- Yeoman - nástroj na automatizované vytváranie nových súborov, špecifiických pre použitý framework (http://yeoman.io/)

Viac informácií o jednotlivých nástrojoch nájdete na vyššie uvedených stránkach projektov.

Tieto závislosti nainštalujete týmto príkazom (podľa nastavenia adresára NPM možno bude potrebné použiť sudo)

```
npm install -g grunt-cli bower yo
```

Ďalej je potrebné nainštalovať yeoman generátor s názvom generator-angular:

```
npm install --global generator-angular
```

Po naklonovaní repozitáru je potrebné lokálne nainštalovať závislosti projektu. Spustite príkazy:

```
npm install
bower install
```

Po vykonaní predchádzajúcich krokov je aplikácia pripravená na spustenie. Na spustenie lokálneho servera, zadajte príkaz:

```
grunt serve
```

Po vykonaní tohto príkazu by sa malo v prehliadači automaticky otvoriť okno s bežiacou aplikáciou.

Upozornenie: nikdy nespúšťajte aplikáciu priamo z app/index.html, ale vždy spustite lokálny server.

Príprava pred vývojom

Po stiahnutí najnovšej verzie z repozitára je vždy potrebné spustiť príkazy.

```
npm update
bower update
```

Tieto príkazy nainštalujú, alebo aktualizujú, moduly, ktoré do projektu pridal iný člen tímu, a sú potrebné na beh aplikácie.
Pri tvorbe kódu a následnej kontrole funkcionality v prehliadači, je užitočná nasledujúce úloha: (spustite v inom tabe alebo okne terminálu, ako samostatný proces):

```
grunt watch
```

Táto úloha, pri každej zmene zdrojového kódu, spôsobí kompiláciu a automatický refresh otvorenej aplikácie, s aktualizovanými zdrojovými kódmi.

Veľmi dôležitá je úloha *js:hint*, ktorá spustí kontrolu javascript kódu. Každý pull-request s nenulovým počtom chýb bude automaticky zamietnutý.

Pridávanie modulov tretej strany

Kvôli ľahšiemu managementu modulov tretej strany a ich verzií (modul na vykresľovanie grafov, frameworky) je nutné, pokiaľ je to možné, ich pridávať ako bower moduly. Viac informácií o tom, ako používať bower, nájdete tu (http://bower.io/). Informáciu o názve bower modulu je väčšinou možné nájsť na stránkach projektu daného modulu, alebo pomocou príkazu:

```
bower search <nazov_modulu>
```

Na inštaláciu modulu do projektu zadajte (nezabudnite byť nastavený v priečinku projektu):

```
bower install <nazov_modulu> --save
```

Upozornenie: je dôležité použiť prepínač --save, v opačnom prípade by sa modul neuložil do zoznamu modulov v bower.json, a ostaným členom tímu by sa tento modul nenainštaloval.

Aby sa súbory modulu (vo väčšine prípadov súbory javascriptu a štýlov) automaticky nalinkovali do projektu (súbor app/index.html), spustite:

```
grunt wiredep
```

Odteraz môžete používať funkcie nainštalovaného modulu.

Pridávanie funkcionality do aplikácie

Aplikácia je napísaná vo frameworku AngularJS. Vytváranie controllerov, služieb a views je zautomatizované pomocou nástroja Yeoman. Projekt obsahuje angular generátor (https://github.com/yeoman/generator-angular), preto je možné generovať Angular špecifické súbory pomocou príkazov:

```
yo angular:controller myController
yo angular:directive myDirective
yo angular:filter myFilter
yo angular:service myService
```

Tieto príkazy vygenerujú súbory s príslušným komponentom, a takisto automaticky vygenerujú spec súbory potrebné na unit testy (východzí je Jasmine). Tie sa nachádzajú v zložke *test/spec*. Unit testy spustíte príkazom:

```
grunt test
```

Ku každému vytvorenému controlleru/službe/filtru je potrebné napísať unit test. Každá použitá premenná a metóda, ktorú je obsiahnutá v *scope,* musí mať napísaný aspoň jeden test pri inicializácií, a jeden pri zmene. Framework, ktorý momentálne používame na písanie testov je Jasmine (http://jasmine.github.io/). V prípade, ak je vhodnejšie použiť iný framework, kontaktujte ostatných členov tímu a po vzájomnej dohode je možné používať viacero frameworkov.

Build aplikácie

Build aplikácie nie je nutné vykonávať, vykonáva sa len pri deployovaní na server.

```
grunt build
```

Build aplikácie sa nachádza v priečniku *dist.*

V prípade akýchkoľvek nejasností pri postupoch vyvíjania, prosím kontaktujte @PatrikGallik prostredníctvom HipChatu.

9 Záznamy zo stretnutí

Kompletný zoznam zápisníc zo stretnutí 1 až 7.

Zápis z tímového stretnutia č.1

Dátum:	2.10.2014		
Miesto stretnutia:	Jobsovo softvérové štúdio		
Čas:	10.00 – 11.00		
Účastníci:	Vedúci projektu:	Ing. Peter Lacko, PhD.	
	Členovia tímu:	Bc. Juraj Kochjar	Bc. Martin Kaššay
		Bc. Matej Kloska	Bc. Patrik Gallik
		Bc. Pavol Čurilla	Bc. Roman Roštár
Vypracoval:	Bc. Juraj Kochjar		

Téma stretnutia: Oboznámenie sa so súčasným stavom webového rozhrania projektu BOINC@FIIT , organizačno-technický úvod.

Priebeh stretnutia:

- Zoznámenie sa s Jurajom Petríkom – minuloročným participantom na projekte BOINC@FIIT, konzultácie ohľadom implementačných detailov existujúceho rozhrania, Juraj poskytol zdrojové súbory potenciálnych riešení vzhľadu rozhrania.
- Zoznámenie sa s vedúcim projektu
- Demonštrácia súčasného stavu webového rozhrania a diskusia k nemu

Zadané úlohy do budúceho stretnutia:

Popis úlohy	Adresát
Stanoviť si prvotné rozloženie funkcií v tíme	Celý tím
Vytvoriť šablónu pre budúce zápisy aj dokumentáciu projektu	Manažér dokumentovania
Založiť projektový denník	Celý tím
Analyzovať iné nástroje riešiace vzhľad webového rozhrania, iné ako nám poskytol Juraj Petrík, zároveň pre každý uviesť argumenty za a proti jeho použitiu	Celý tím

Zhodnotenie stretnutia: Prvé stretnutie s vedúcim slúžilo na úvodné oboznámenie sa so súčasným stavom webového rozhrania pre distribuované výpočty. Oboznámili sme sa so základnými princípmi fungovania distribuovaných výpočtov. Po stretnutí sme si ihneď stanovili tímové roly, a to nasledovne:

Vedúci tímu:	Bc. Roman Roštár
Hlavný architekt:	Bc. Patrik Gallik
Manažér kvality:	Bc. Martin Kaššay
Manažér dokumentovania:	Bc. Juraj Kochjar
Manažér podporných prostriedkov:	Bc. Pavol Čurilla
Manažér vývoja:	Bc. Matej Kloska

Zápis z tímového stretnutia č.2

Dátum:	9.10.2014		
Miesto stretnutia:	Jobsovo softvérové štúdio		
Čas:	10:00 – 12:00		
Účastníci:	Vedúci projektu:	Ing. Peter Lacko, PhD.	
	Členovia tímu:	Bc. Juraj Kochjar	Bc. Martin Kaššay
		Bc. Matej Kloska	Bc. Patrik Gallik
		Bc. Pavol Čurilla	Bc. Roman Roštár
Vypracoval:	Bc. Pavol Čurilla		

Téma stretnutia: Ujasnenie si základnych predstáv o projekte a upresnenie si spôsobov riadenia projektu.

Priebeh stretnutia:

- Vyhodnotenie úloh zadaných na predošlom tímovom stretnutí.
  - Funkcie v tíme rozdelené boli.
  - Šablóna zápisnice vytvorená.
  - Projektový denník bude osobnou zodpovednosťou každého člena tímu.
  - Licencia k nástroju Jira bola získaná zdarma.
- Diskusia k:
  - Použitiu iných nástrojov riešiace vzhľad webového rozhrania projektu.
  - Výzoru landing page.
  - Spôsobu riešenia backlogu.
  - Ošetreniu veľkého počtu falošných používateľov.

Zadané úlohy do budúceho stretnutia:

Popis úlohy	Adresát
Prihlásenie tímu do TP cupu	Bc. Juraj Kochjar
Spustenie Boinc servra na VM	Bc. Matej Kloska
Vytvorenie skriptu, ktory zmaže mesiac neaktívnych používateľov	Bc. Martin Kaššay
Vytvorenie tímového webu	Bc. Patrik Gallik
Zavedenie CAPTCHA pri registracii užívateľov	Bc. Pavol Čurilla

Vytvorenie repozitára pre projekt	-

Zhodnotenie stretnutia: Druhé stretnutie tímu slúžilo k ujasneniu si základných predstáv o projekte, upresneniu spôsobu riadenia projektu a návrhu riešení existujúceho problému s falošnými používateľmi. Otvorená ostáva otázka použitia nástrojov riešiacich vzhľad webového rozhrania, je potrebné bližšie preštudovať zdrojové kódy a dohodnúť sa na ďalšom postupe. Ako backlog projektu bude používaný nástroj JIRA, každý člen tímu bude môcť pridať nové úlohy. Je potrebné aktívne sledovanie webu TP a prednášok. Za Scrum Mastera bol určený **Bc. Roman Roštár**.

Zápis z tímového stretnutia č.3

Dátum:	16.10.2014		
Miesto stretnutia:	Jobsovo softvérové štúdio		
Čas:	10:00 – 12:00		
Účastníci:	Vedúci projektu:	Ing. Peter Lacko, PhD.	
	Členovia tímu:	Bc. Juraj Kochjar	Bc. Martin Kaššay
		Bc. Matej Kloska	Bc. Patrik Gallik
		Bc. Pavol Čurilla	Bc. Roman Roštár
Vypracoval:	Bc. Pavol Čurilla		

Téma stretnutia: Diskusia na tému možností ďalšieho postupu riešenia

Priebeh stretnutia:

- Vyhodnotenie úloh zadaných na predošlom tímovom stretnutí.
  - Na serveri je spustený BOINC
  - Bol vytvorený tímový web
  - Bol vytvorený repozitár pre verziovanie zdrojových kódov
  - Pre implementáciu captcha zabezpečenia registrácie sa použije knižnica reCAPTCHA
  - Zápis tímu na TP Cup bol odložený na ďalší týždeň
  - Vytvorenie skriptu pre mazanie neaktívnych používateľov je v procese riešenia, odkladá sa na budúci týždeň

- Diskusia k:
  - Voľbe technológie pre riešenie projektu
  - Architektúra riešenia
  - Organizáciu v tíme

Zadané úlohy do budúceho stretnutia:

Popis úlohy	Adresát
Prihlásenie tímu do TP cupu	Bc. Juraj Kochjar
Vytvorenie hookova prepojenie služieb na repozitáre, organizácia nástroja JIRA	Bc. Matej Kloska
Vytvorenie skriptu, ktory zmaže mesiac neaktívnych používateľov	Bc. Martin Kaššay
Úpravy tímového webu, vytvorenie skeletu SPA	Bc. Patrik Gallik

Zavedenie CAPTCHA pri registracii užívateľov	Bc. Pavol Čurilla
Vytvorenie Ruby on Rails aplikácie	Bc. Roman Roštár

Zhodnotenie stretnutia: Na treťom stretnutí sa riešili otázky ohľadne analýzy aktuálneho stavu riešenia a výberu správnej technológie pre ďalšie pokračovanie. Hodnotili sme klady a zápory rozličných možností riešenia tohto projektu. Hlavným problémom bol neprehľadný a úzko previazaný „spaghetti“ kód aktuálneho riešenia, ktorý navyše absentuje testy. Z tohto dôvodu sme sa rozhodli zvoliť nasledovné riešenie:

- Frontend ako single pageapplication (vo frameworku Angular.js/React.js)
- API komunikujúce s frontendovou časťou vo formáte json

Pre tvorbu API bol vybraný framework Ruby on Rails. Časti aktuálneho riešenia, ktoré by nebolo možné namapovať v tomto frameworku bude nutné prispôsobiť z aktuálneho riešenia v jazyku PHP.

Zápis z tímového stretnutia č.4

Dátum: 23.10.2014
Miesto stretnutia: Jobsovo softvérové štúdio
Čas: 10.00 – 13.30
Účastníci: Vedúci projektu: Ing. Peter Lacko, PhD.
Členovia tímu: Bc. Juraj Kochjar, Bc. Martin Kaššay, Bc. Matej Kloska, Bc. Patrik Gallik, Bc. Pavol Čurilla, Bc. Roman Roštár
Vypracoval: Bc. Patrik Gallik

Téma stretnutia: Kick-start implementácie projektu
Priebeh stretnutia:

- Ukončili sme všetky doterajšie úlohy v Jire a začali sme nový šprint
- Dohodli sme sa na používaní nástroja HipChat na komunikáciu
- Dohodli sme sa na používaní moqups.com ako prototypovacieho nástroja
- Diskutovali sme prvý prototyp landing page
- Vytvorili sme architektonický návrh aplikácie - na tabuli
- Vytvorili sme user stories a na ich základe povytvárali tasky v Jire do nasledujúceho šprintu

Zadané úlohy do budúceho stretnutia:

Popis úlohy	Adresát
Nový šprint v Jire, podrobný rozpis úloh nájdeme tam	Celý tím
Vytvorenie prihlášky do TP cupu	Celý tím

Zhodnotenie stretnutia: Výsledkom nášho stretnutia bolo uzavretie predošlého šprintu a vytvorenie nového. Uzavreli sme fázu analýzy a pustili sme sa do implementácie. Vytvorili sme tasky v Jire pomocou rozdelenia na user stories. Potom sme tieto user stories ohodnotili story pointmi - pomocou peknej mobilnej aplikácie Scrum Time.

Vedúci tímu:	Bc. Roman Roštár
Hlavný architekt:	Bc. Patrik Gallik
Manažér kvality:	Bc. Martin Kaššay
Manažér dokumentovania:	Bc. Juraj Kochjar
Manažér podporných prostriedkov:	Bc. Pavol Čurilla
Manažér vývoja:	Bc. Matej Kloska

Zápis z tímového stretnutia č.5

Dátum:	30.10.2014		
Miesto stretnutia:	Jobsovo softvérové štúdio		
Čas:	10.00 – 13.30		
Účastníci:	Vedúci projektu:	Ing. Peter Lacko, PhD.	
	Členovia tímu:	Bc. Juraj Kochjar	Bc. Martin Kaššay
		Bc. Matej Kloska	Bc. Patrik Gallik
		Bc. Pavol Čurilla	Bc. Roman Roštár
Vypracoval:	Bc. Matej Kloska		

Téma stretnutia: Priebežné stretnutie k druhému šprintu.
Priebeh stretnutia:

- Prebrali sme chýbajúce nutné časti, ktoré potrebujeme dodefinovať:
    - codereview,
    - dokumentovanie – automatizovaná dokumentácia + špecifická dokumentácia, ktorá sa nachádza pri každom projekte na Bitbuckete,
- Prechod z ručne písanej zápisnice na automatizovanú – export z nástroja JIRA,
- Dohodli sme sa na podrobnom a priebežnom komentovaní všetkých úloh v systéme,
- Vyhodnotili sme spätne prihlášku na TPCUP a dohodli sa na panelovej diskusii – rozhodli sme sa na panelovej diskusii ohľadom prezentovania do zimného semestra.
- Odsúhlasili sme základné návrhy.
- Rozdelenie metodologii:
    - Roman: GIT branching
    - Juraj: ako písať štýly v SaSS
    - Patrik: JavaScript / AngularJS codestyle
    - Martin: codereview proces
    - Matej: proces špecifikácie REST komunikácie
    - Pavol: postup pre definovanie úloh v JIRE

- Musíme vyhodnotiť minuloročnú dokumentáciu tímu.
- Potrebujeme vytvoriť vzorovú aplikáciu pre potreby otestovania zadávania úlohy.

Zadané úlohy do budúceho stretnutia:

Popis úlohy	Adresát
Nový šprint v Jire, podrobný rozpis úloh nájdeme tam	Celý tím
Vytvorenie prihlášky do TP cupu	Celý tím

Zhodnotenie stretnutia: Výsledkom nášho stretnutia bolo uzavretie predošlého šprintu a vytvorenie nového. Uzavreli sme fázu analýzy a pustili sme sa do implementácie. Vytvorili sme tasky v Jire pomocou rozdelenia na user stories. Potom sme tieto user stories ohodnotili story pointmi - pomocou peknej mobilnej aplikácie Scrum Time.

Vedúci tímu: Bc. Roman Roštár
Hlavný architekt: Bc. Patrik Gallik
Manažér kvality: Bc. Martin Kaššay
Manažér dokumentovania: Bc. Juraj Kochjar
Manažér podporných prostriedkov: Bc. Pavol Čurilla
Manažér vývoja: Bc. Matej Kloska

Zápis z tímového stretnutia č.6

Dátum:	6.11.2014		
Miesto stretnutia:	Jobsovo softvérové štúdio		
Čas:	10.00 – 13.30		
Účastníci:	Vedúci projektu:	Ing. Peter Lacko, PhD.	
	Členovia tímu:	Bc. Juraj Kochjar	Bc. Martin Kaššay
		Bc. Matej Kloska	Bc. Patrik Gallik
		Bc. Pavol Čurilla	Bc. Roman Roštár
Vypracoval:	Bc. Martin Kaššay		

Téma stretnutia: Stretnutie ku koncu druhého šprintu

Priebeh stretnutia:

- Dohodli sme sa na používaní bitbucket pre codereview.
- Set-up codeship.
- Prediskutovanie copywrite textov.
- Pull request pre landing page.
- Dohodnutý Mandrill pre transakčné maily.

Zadané úlohy do budúceho stretnutia:

Popis úlohy	Adresát
Dokončenie šprintu a štart nového	Celý tím
Dokončenie JSON komunikácie pre prihlásenie a registráciu	Patrik Gallik
Vloženie dummy úlohy a návrh user pages apps	Juraj Kochjar
Konfigurácia dohodnutého transakčného mailu do RoR	Roman Roštár
Integrácia LDAP pre prihlásenie user-a	Matej Kloska

Zhodnotenie stretnutia: Výsledkom nášho stretnutia bolo dohodnutie konca šprintu. Rozdelenie úloh pre jeho dokončenie. Dohodnutie tretieho šprintu v priebehu najbližšieho víkendu.

Vedúci tímu:	Bc. Roman Roštár
Hlavný architekt:	Bc. Patrik Gallik
Manažér kvality:	Bc. Martin Kaššay
Manažér dokumentovania:	Bc. Juraj Kochjar
Manažér podporných prostriedkov:	Bc. Pavol Čurilla
Manažér vývoja:	Bc. Matej Kloska

Zápis z tímového stretnutia č.7

Dátum: 13.11.2014
Miesto stretnutia: Jobsovo softvérové štúdio
Čas: 10.00 – 13.30
Účastníci: Vedúci projektu: Ing. Peter Lacko, PhD.
Členovia tímu: Bc. Juraj Kochjar Bc. Martin Kaššay
Bc. Matej Kloska Bc. Patrik Gallik
Bc. Pavol Čurilla Bc. Roman Roštár
Vypracoval: Bc. Juraj Kochjar

Téma stretnutia: Stretnutie k úvodu tretieho šprintu
Priebeh stretnutia:

- Na začiatku stretnutia prebehol skupinový stand-up, pri ktorom každý z členov tímu prezentoval svoju súčasnú prácu a problémy, ktoré sa vyskytli počas nej. Uviedli sme Petra do problematiky.
- Mali sme problém pripojiť sa na server ucebne.fiit.stuba.sk z lokálnej siete, Peter problém vyriešil.
- Migrácia databáz je funkčná spolu s modelmi dát.
- Autentifikácia prihlásenia sa z AIS pomocou Eldap protokolu bude prebiehať interne v našej Ruby aplikácii, ktorá nutne musí byť spúšťaná na serveri s fakultnou IP adresou.
- Príprava a diskusia ohľadom panelovej prezentácie.
- Konfigurácia nástroja Mandrill

Zadané úlohy do budúceho stretnutia:

Popis úlohy	Adresát
Funkčná demo verzia produktu do najbližšieho stretnutia	Celý tím

Zhodnotenie stretnutia: Výsledkom nášho stretnutia bolo oboznámenie vedúceho tímu so súčasným stavom prác na projekte, stanovenie si priorít a úloh do tetieho šprintu.

Vedúci tímu:	Bc. Roman Roštár
Hlavný architekt:	Bc. Patrik Gallik
Manažér kvality:	Bc. Martin Kaššay
Manažér dokumentovania:	Bc. Juraj Kochjar
Manažér podporných prostriedkov:	Bc. Pavol Čurilla
Manažér vývoja:	Bc. Matej Kloska