Metodika implementácie

Tabuľka verzií

DÁTUM	VERZIA	POPIS ÚPRAV	ZODPOVEDNÁ OSOBA
16.11.2014	1.0	Vytvorenie prvej verzie dokumentu	Martin Tamajka

Obsah

1 Metadata

Vymedzenie obsahu

Tento dokument opisuje metodiku implementácie a všetky jej najdôležitejšie aspekty.

V úvode metodika vymedzuje programovacie jazyky, ktoré budú používané v rámci projektu, a taktiež uvádza ich základné využitie. Následne sú uvedené všeobecné zásady, ktoré mali významný vplyv na celú metodiku.

V ďalších častiach sa metodika postupne zaoberá konvenciou umiestňovania súborov a jednotlivých častí aplikácie v rámci frameworku Ruby on Rails. Dôraz je pritom kladený najmä na súbory, ktorých umiestnenie nie je v rámci ROR ustálené, a ohľadom ktorých by mohli vzniknúť nejasnosti. Následne sú taxatívne vymenované najdôležitejšie oblasti využitia jednotlivých SW prostriedkov / jazykov, a to ako na strane serveru, tak na strane klienta. Ďalej metodika ošetruje pomenúvanie súborov.

Jednou z najdôležitejších častí je definovanie tzv. "Coding conventions".

V poslednej časti je uvedený spôsob, akým je nutné vykonávať review zdrojových kódov a k nim pridružených úloh.

Dedikácia metodiky

Tvorcom tejto metodiky je jej autor, Martin Tamajka. Metodika bola zároveň aktívne diskutovaná najmä s Ľubomírom Vnenkom, nakoľko tento má najväčšie skúsenosti s FW Ruby on Rails.

Cieľom metodiky je zabezpečiť dodržiavanie zásad čistého programovania a konvencií ROR. Taktiež ňou chceme zosumarizovať všetky zásady, ktoré by mal každý z vývojárov pri vývoji dodržiavať, čím sa dosiahne jednotnosť vyprodukovaného kódu, a teda bude ľahšie sa v ňom zorientovať.

Dôležitou časťou je taktiež ošetrenie spôsobu revidovania (review) úloh. Týmto chceme dosiahnuť vyššiu kvalitu konečného produktu.

Vymedzenie nástrojov

Nástrojmi používanými pre metodiku sú najmä samotné vývojové prostredie (toto nie je striktne dané, záleží len od preferencie vývojára), framework ROR a služby poskytované serverom GitBucket.

Nadväzujúce dokumenty a metodiky

S týmto dokumentom priamo súvisia metodiky:

- Dokumentácia riadenia projektu
- Dokumentácia k inžinierskemu dielu
- Metodika evidencie úloh v Redmine systéme
- Metodika GIT

2 Softvérové prostriedky

Strana serveru

- Aplikačný kód použitý na serverovej strane bude **Ruby**. Ako framework bude použitý **Ruby on Rails** s príslušnými **GEM**-mi.
- Aplikačným serverom využívaným Ruby on Rails bude **Apache**

Strana klienta

- Na definíciu vzhľadu kódu HTML bude použitý **CSS** framework **SASS**
- Jazykom zabezpečujúcim interakcie, validácie, dynamickosť a synchrónnu a asynchrónnu komunikáciu na strane klienta bude **Javascript** a jeho nadstavby **JQuery** a **CoffeeScript**

3 Všeobecné zásady vývoja

Metodika je vytvorená v súlade so zásadami **DRY** (Don't repeat yourself) a **CoC** (Convention over configuration).

DRY

Všetko v rámci projektu má jasne určené miesto a jasne určené pomenovanie, čo má za cieľ zaručiť, že v rámci projektu sa nebudú vyskytovať duplicity (typicky duplicity v kóde, prípadne duplicitne udržiavané súbory).

CoC

Cieľom je mať v rámci frameworku (prípadne jeho rozšírenia) jasne určené pravidlá pomenúvania, umiestňovania a konfigurácie častí projektu (kódy, súbory, konfiguračné súbory a iné). Toto umožňuje jednoduchšiu orientáciu v projekte a použitie automatizovaných postupov a nástrojov (napr. prepojenie modelov a databázy na základe mennej konvencie v ROR či prípadne umiestňovanie "assets").

4 Vytváranie a umiestňovanie nových súčastí projektu

Ruby on Rails

Na vytváranie **modelov** a **controllerov** sa používa **scaffolding**. Zabezpečí sa tým ich umiestnenie v rámci konvencií ROR (tieto konvencie sú ľahko dostupné a všeobecne známe, preto ich tu nebudeme uvádzať).

Model

- V prípade, že je potrebné generovať model bez uvedenia jeho atribútov, použije sa nasledovná forma príkazu:
  - *rails generate model NazovModelu*
- V prípade, že je potrebné generovať model s uvedením atribútov:
  - *rails generate model NazovModelu nazov_atributu:typ_atributu nazov_atributu2:typ_atributu ... nazov_atributuN:typ_atributu*

Model je vždy generovaný s prepínačom --migration nastaveným na hodnotu true, nakoľko *všetky zmeny v databáze budú vykonávané prostredníctvom migrácií*, a teda nie priamym zásahom do databázy. Tento atribút nie je nutné používať, keďže true je preddefinovanou hodnotou tohto prepínača.

Po vygenerovaní modelu je nutné do súboru seeds.rb doplniť seedovacie dáta pre tento konkrétny model. Minimálny počet záznamov je pre testovacie účely 20.

Controller

Controllery sú generované vždy automatizovaným spôsobom. Cieľom tohto je dosiahnuť:

- jednotnú formu controllerov,
- zabrániť problémom so smerovaním (v prípade zabudnutia definovania smerovania v súbore routes.rb),
- vygenerovať štandardné súbory spojené s controllerom (helpery, testy, assety) a štandardnú adresárovú štruktúru pre daný controller. Takto vygenerované súbory budú v projekte zahrnuté, aj keď budú v danom okamihu prázdne. Cieľom

je dosiahnuť konzistentnosť v projekte a ľahké doplnenie funkcionality v budúcnosti bez nutnosti vytvárať ďalšie súbory ručne.

Controllery musia mať už pri generovaní definované akcie (inak by controllery nemali zmysel. Samozrejme, je možné v budúcnosti doplniť ďalšie akcie do controlleru). Na samotné generovanie bude použitý príkaz:

generate controller ControllerName action_name1 action_name2 … action_nameN

Všeobecný zdroj (resource)

V prípade nutnosti generovať zdroj so všetkými CRUD akciami (napríklad používateľa, ktorého je nutné umožniť vytvoriť, upraviť, aktualizovať a vymazať), bude použitá všeobecná forma scaffoldu poskytovaného ROR. Tento zabezpečí vytvorenie modelu, controlleru, migračného skriptu, testov, kostier potrebných views, assetov a helperov. Tiež zabezpečí nastavenie korektného smerovania v routes.rb.
Vytvorenie všeobecného zdroja sa dosahuje použitím nasledujúceho príkazu:

generate scaffold ModelName nazov_atributu1:typ_atributu … nazov_atributuN:typ_atributu

Obrázky

Vlastné obrázky

Budú umiestňované do adresára **app/assets/images/.**

Podrobnejšie členenie:

- Obrázky špecifické pre views konkrétneho controlleru budú umiestňované do adresára **app/assets/images/controller_name/**
- Ikony budú umiestňované do adresára **app/assets/images/icons/**
- Obrázky všeobecne využívané v rámci celej aplikácie budú umiestňované do adresára **app/assets/images/common/** (napr. logo)

Prevzaté obrázky

Budú umiestňované do adresára **vendor/assets/images**/

- Obrázky špecifické pre views konkrétneho controlleru budú umiestňované do adresára vendor/**assets/images/controller_name/**

CSS a SASS

Vzhľad dokumentov bude kaskádovými štýlmi definovaný v súboroch štandardne vytvorených a pridružených za pomoci scaffold systému poskytovaného ROR. Výnimkou sú prevzaté kaskádové štýly, ktoré budú umiestňované do adresára **vendor/assets/stylesheets/.**

Ak existuje CSS a SASS tretej strany vo forme GEMu, bude použité toto. V opačnom prípade bude použité priamo CSS a SASS a toto bude umiestnené do adresára **/vendor/assets/stylesheets/.**

Javascript

Podobne ako pri CSS, vlastný javascript bude umiestňovaný do štandardných súborov vytvorených ROR. Je tu však zásadný rozdiel - keďže v javascripte budú používané ako CoffeeScript, tak JQuery, je možné v priečinku **app/assets/javascripts/** okrem súboru **jsfilename.js.coffee** aj súbor **jsfilename_jq.js**, ktorý bude obsahovať kód javascriptu, resp. JQuery. Vždy je možné pripojiť k názvu súboru príponu **.erb**, ak je nutné použiť v rámci javascriptu ruby.

Ak existuje Javascript tretej strany vo forme GEMu, bude použitý tento. V opačnom prípade Javascripty tretích strán budú umiestňované do adresáta **vendor/assets/javascripts.**

Súbory nahrané používateľom

- Súbory nahrané na server používateľom (typicky profilové fotky a pod.) sa umiestňujú do adresára **public/uploads**
- Špeciálne obrázky nahrané používateľom na server (typicky profilové fotky a pod.) sa umiestňujú do adresára **public/uploads/images**

Upozornenie

Žiadny view nesmie priamo obsahovať definíciu kaskádových štýlov alebo javascriptu. Tieto musia byť vždy pripojené len prostredníctvom externých súborov.

5 Spôsob použitia jednotlivých jazykov

Strana serveru

Aplikačný kód na strane serveru bude tvorený **ROR**. Na strane serveru budú vykonávané:

- Autentifikácia používateľa
- Verifikácia údajov odoslaných používateľom prostredníctvom formulárov
- Spojenie a práca s databázou
- Logovanie
- REST-ové webové služby
- Výkon aplikačnej logiky

Strana klienta

Vzhľad views bude definovaný prostredníctvom kaskádových štýlov, konkrétne prostredníctvom rozšírenia **SASS**.

Za pomoci javascriptu (JS, Coffee, JQuery) budú vykonávané:

- validácia formulárových dát
- komunikáciu so serverom prostredníctvom synchrónnych a asynchrónnych volaní technikou AJAX
- tvorba animácií a dynamického charakteru views
- nahrávanie súborov na server

6 Konvencia pomenúvania súborov

- Ruby on Rails
  - Riadi sa štandardmi Ruby on Rails
- javascript, CoffeeScript, JQuery
  - Riadi sa štandardmi Ruby on Rails
  - V prípade potreby vytvoriť súbor iný než aké súštandardné pre ROR, tieto budú umiestnené do adresára **app/assets/javascripts/special**. Názov takýchto súborov bude vo forme:
    - **problemName-version.js | problemName-version.js.coffee | problemName-version_jq.js**
    - každý z predchodzích názvov môže pribrať koncovku .erb, ak je nutné použiť v rámci javascriptu ruby
- CSS, SASS
- Riadi sa výlučne štandardmi Ruby on Rails
- Obrázky
  - Špecifické pre používateľa: **idpouzivatela_imgname.imgextension**
  - Ostatné obrázky nemajú špecifické názvoslovie

Dodatok k Javascriptu:

"version" v názve súboru je desatinné číslo v tvare

cislo_pred_desatinnou_bodkou.cislo_za_desatinnou_bodkou (napr. 2.3),

pričom prvá verzia javascriptového súboru je rovná 1.0. Následne je verzia menená nasledovne:

- po pridaní novej funkcionality zvýš cislo_pred_desatinnou_bodkou o 1 a nastav cislo_za_desatinnou_bodkou na 0,
- po úprave (oprava / rozšírenie / refaktorizácia) už existujúcej funkcionality zvýš cislo_za_desatinnou_bodkou.

7 Kód

Všeobecné zásady čistoty kódu

- Nepoužívať tzv. bulharské konštantny - ak má hodnota svoju sémantiku (napr. číslo 3.14…), je potrebné priradiť ho do premennej, resp. konštanty a následne používať túto.
- Často sa opakujúca parametrizovateľná sekvencia príkazov (napr. overenie stavu relácie ("session") kvôli autentifikácií používateľa) musí byť vyčlenené ako samostatná metóda. Toto sa netýka neparametrizovateľnej sekvencie (napr. výber z databázy a filtrácia podľa rôznych parametrov).

Písanie komentárov

- Komentáre sú písané anglicky.
- Je nutné (ak už je potrebné komentovať) komentovať to, čo kód robí, a nie ako to robí (to je možné ľahko vyčítať zo samotnej syntaxe).
- Nad každou metódou musí byť komentár, ktorý deklaratívne v prítomnom čase opisuje činnosť danej metódy. Je potrebné opísať vstupy a výstupy. Ak je to nutné, komentár obsahuje aj vysvetlenie významu jednotlivých argumentov metódy. Napr:

#Input: array of players

#Output: array of pairs of players that play together

#Pair players depending on their scores with respect to Dutch system

def apply_dutch players

...

- Na formátovanie komentárov sa použije RDoc
- Controllery
  - Pre každú akciu controlleru je nutné v komentári definovať, na akú požiadavku daná akcia odpovedá (GET, POST, …), akými formami je akcia schopná

odpovedať (html, json, …) a ako vyzerá URL, za ktorou sa skrýva daná akcia (štandardný komentár generovaný scaffoldom ROR). Počet riadkov komentára je rovný počtu rôznych foriem odpovede akcie. Každý riadok má pritom formu # request /url[.answer], kde v prípade, že odpoveďou je html, uvedenie formy odpovede vynechávame. Napr:

 # POST /players

 # POST /players.json

def create

...

- Pri zložitejších interakciách s databázou (komplexné dopyty prostredníctvom ActiveRecord) je potrebné komentárom objasniť výsledok takejto interakcie. Príkladom je zložitá filtrácia záznamov z DB. Takýto komentár pridávame do riadku za daný príkaz, pričom takýto komentár neobjasňuje, ako sú dáta filtrované, ale aké sú výsledné dáta. Napr:

@players = Player.where(age: 55, hair: 'none') #old hairless players

~~*@players = Player.where(age: 55, hair: 'none') #players with age=55 and with hair='none'*~~

- Modely
  - V rámci modelov sú metódy komentované vyššie opísanou technikou.
  - V rámci riadkov sú komentáre umiestňované podľa uváženia programátora. Takéto komentáre opisujú výsledok činnosti daného kódu, nie kód samotný
- Javascript
  - V rámci Javascriptu sú funkcie a metódy komentované vyššie opísanou technikou.
  - V rámci riadkov sú komentáre umiestňované podľa uváženia programátora. Takéto komentáre opisujú výsledok činnosti daného kódu, nie kód samotný

Čo robiť

- Pridať stručný komentár ku každej metóde na základe vyššie opísaných pravidiel.

Čo NErobiť

- Do zdrojového súboru nepridávať hlavičku vo forme komentára, ktorá obsahuje autora súboru a pod. Tieto informácie sú ľahko dohľadateľné v GITe.
- Nekomentovať bežné programatické operácie (komentáre typu "increase value of number_of_players by 1") v rámci riadku.

8 Validácia údajov od používateľa

- Údaje zadané používateľom sú vo väčšine prípadov vkladané do databázy. Kvôli zachovaniu konzistentnosti databázy, všetky dáta zadané používateľom musia byť validované ako na strane klienta, tak na strane serveru

Na strane serveru

- Validácia prebieha na úrovni modelu
  - Definíciou typu jednotlivých atribútov
  - Použitím metódy **validates** a obdobných metód **validates_*** (za použitia rôznych argumentov týchto metód) poskytovaných ROR pre jednotlivé atribúty modelu. Napr.:

```
class Player < ActiveRecord::Base

    validates :name, :length => { :minimum => 5 }

    ...

end
```

Na strane klienta

- Validácia je vykonávaná Javascriptom
- V prípade, že používateľ zadal do niektorého z prvkov formulára (napr. vstupné pole "input field") neadekvátnu hodnotu, je nutné ho na toto okamžite upozorniť zvýraznením príslušného prvku formulára
- V prípade, že sa používateľ pokúsil odoslať nekorektne (alebo neúplne vzhľadom na povinné polia) vyplnený formulár na server, je nutné ho na toto upozorniť a pri každom z nekorektne vyplnených prvkov formulára vypísať upozornenie

9 Code review

Každá novo implementovaná funkcionalita je implementovaná v samostatnej vetve v rámci verziovacieho systému (v tomto prípade GIT). Po ukončení implementácie funkcionality nastupuje code review. Pre každý kód platí, že zaň nezodpovedá iba ten, kto ho implementoval, ale aj ten, kto vykonáva jeho review. Code review v našom prípade nepodliehajú jednotlivé úlohy ("task"), ale celé používateľské príbehy (US). Postup je pritom nasledovný:

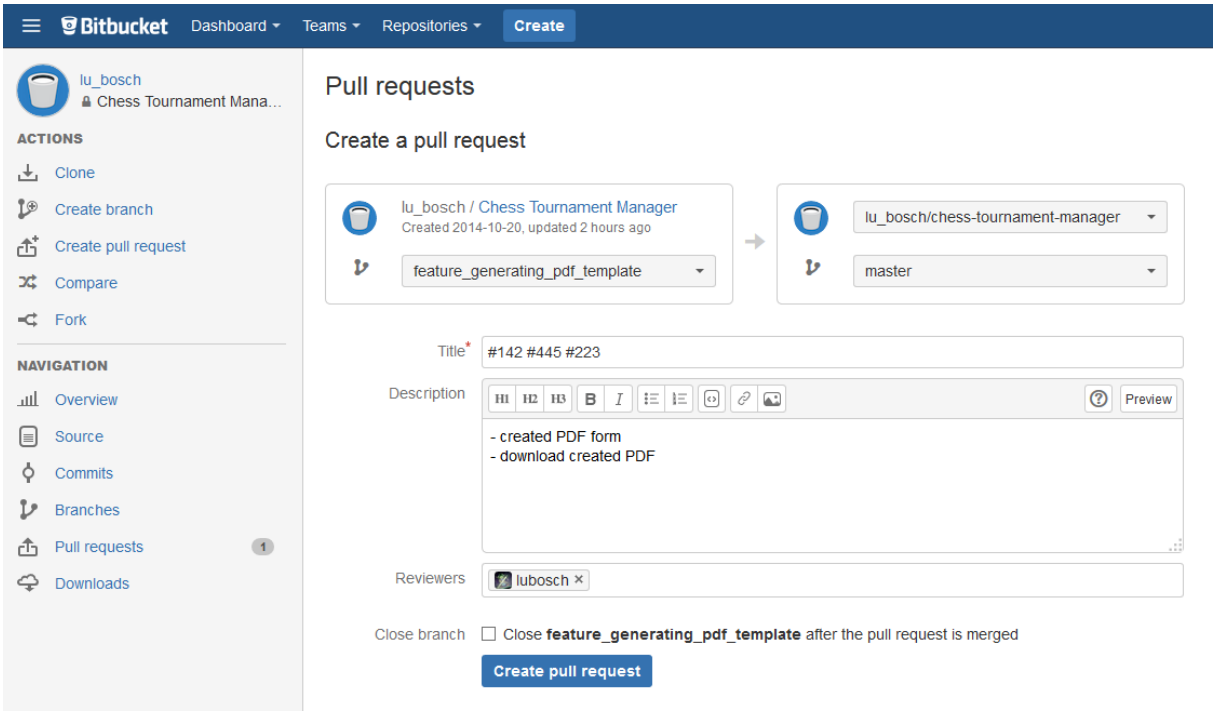
1. Ten, kto funkcionalitu implementuje, ju označí za naimplementovanú, spustí nad touto funkcionalitou automatizované testy. V prípade, že funkcionalita prešla všetkými testami, posunie ju na review reviewerovi.
2. Reviewer prostredníctvom nástroja prezrie implementovanú funkcionalitu (vždy prezerá konkrétny branch). Zameriava sa pritom na:
    a. Funkčnosť daného kódu
    b. Prehľadnosť kódu
    c. Súlad s konvenciami
3. V prípade, že funkcionalita daná na review vyhovuje všetkým trom atribútom kvality, označí reviewer kód ako schválený. V opačnom prípade ku každej nezrovnalosti (*ku konkrétnemu riadku kódu*) pridá komentár s otázkou / konštatovaním ohľadom nezrovnalosti.
4. V prípade, že je na základe code review nutné vytvoriť novú úlohu (bug, logicky chýbajúca funkcionalita, refaktoring), reviewer do komentára vloží na vrchol komentára riadok vo forme:

#ISSUE_NUMBER ; BUG|MISSING_FUNCTIONALITY|REFACTORING ; ISSUE_NAME

Ako nástroj na review slúži **Pull Request** poskytovaný serverom **BitBucket**, ktorý je využívaný ako repozitár pre GIT. V týchto súvislostiach:

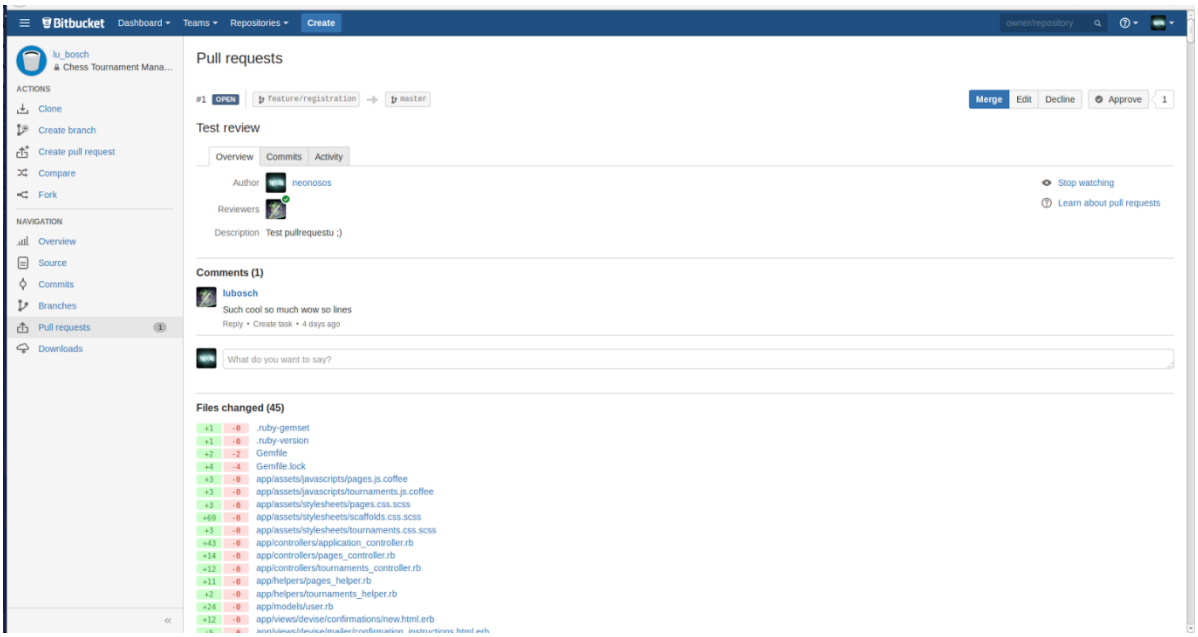
- Posunúť funkcionalitu na review znamená vytvoriť Pull Request v BitBuckete. V rámci vytvárania Pull Requestu je nutné vyplniť (obr. 1):
    - Title: Číslo a opis US, ktorá s implementovanou funkcionalitou súvisí vo forme **#user_story_number user_story_title** , pričom tieto údaje sú získané z Redmine
    - Description: stručný opis toho, čo bolo implementované

- Reviewers: reviewer zodpovedný za review funkcionality
- Branch vľavo hore: branch obsahujúca funkcionalitu, ktorej sa review týka
- Branch vpravo hore: branch, do ktorej má byť reviewovaná branch mergnutá

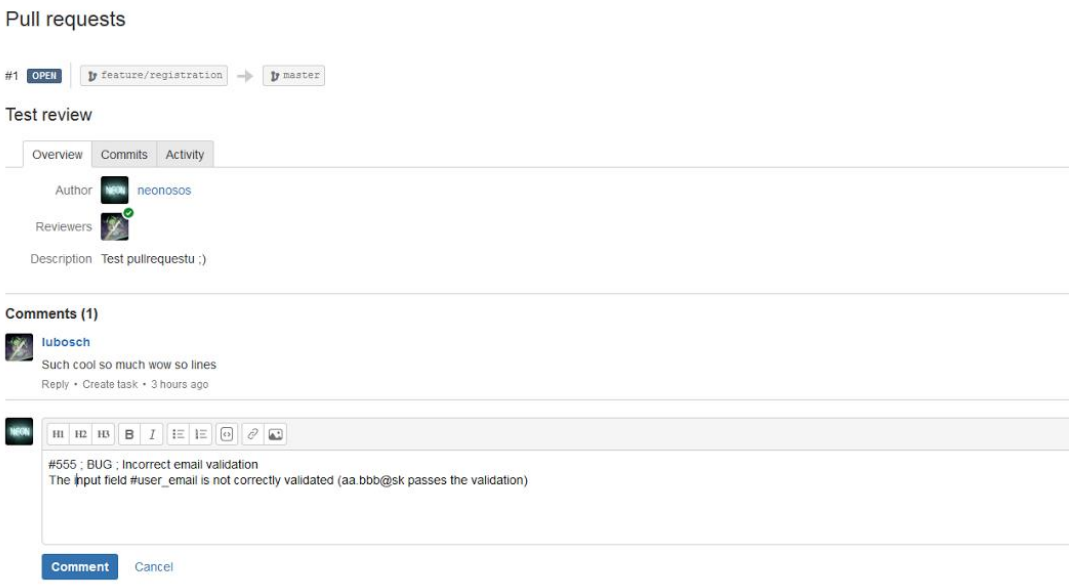


obr. 1

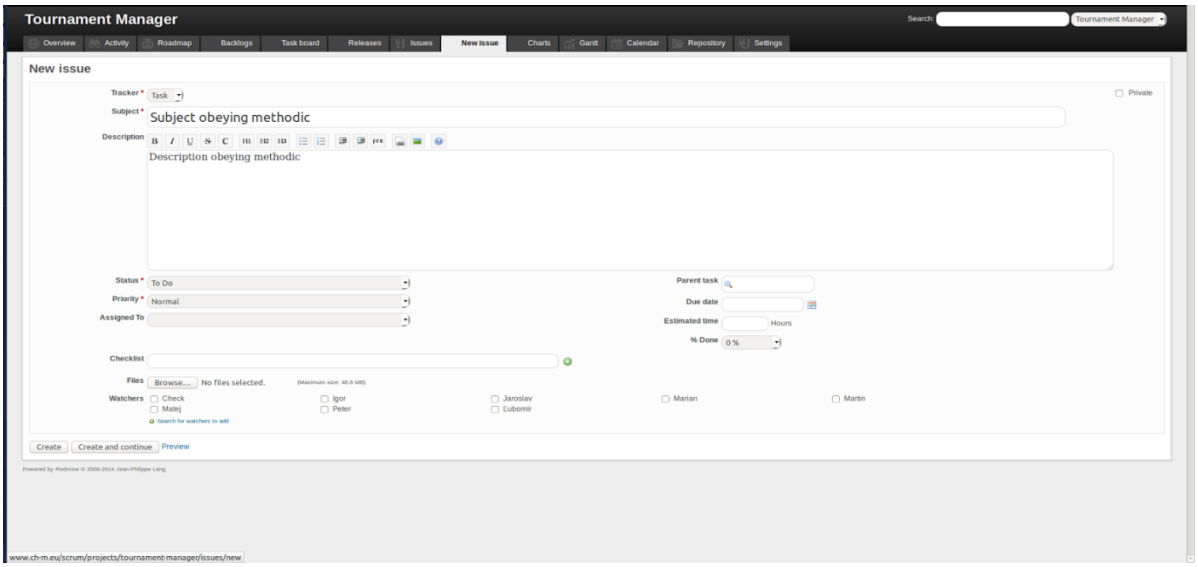
- Schváliť funkcionalitu znamená kliknúť na Approve (obr. 2 vpravo hore)
- Vytvorenie novej úlohy na základe Code Review znamená:
  - Vytvoriť novú úlohu v Redmine (obr. 4). Opis úlohy ako aj jej vytvorenie opisuje príslušná metodika
  - Vyplniť pole "What do you want to say?" (obr. 2)
  - V rámci tohto poľa bude (obr. 3):
    - V prvom riadku číslo novo vytvorenej úlohy (issue v Redmine) podľa metodiky na to určenej
    - V druhom riadku bude stručný opis chyby / chýbajúcej funkcionality/ čo treba refaktorovať


Bitbucket
Dashboard
Teams
Repositories
Create
lu_bosch
Chess Tournament Mana...
ACTIONS
Clone
Create branch
Create pull request
Compare
Fork
NAVIGATION
Overview
Source
Commits
Branches
Pull requests
Downloads
Pull requests
#1 OPEN
feature/registration
master
Merge
Edit
Decline
Approve
Test review
Overview
Commits
Activity
Author
neonosos
Reviewers
Description
Test pullrequestu ;)
Stop watching
Learn about pull requests
Comments (1)
lubosch
Such cool so much wow so lines
Reply • Create task • 4 days ago
What do you want to say?
Files changed (45)
.ruby-gemset
.ruby-version
Gemfile
Gemfile.lock
app/assets/javascripts/pages.js.coffee
app/assets/javascripts/tournaments.js.coffee
app/assets/stylesheets/pages.css.scss
app/assets/stylesheets/scaffolds.css.scss
app/assets/stylesheets/tournaments.css.scss
app/controllers/application_controller.rb
app/controllers/pages_controller.rb
app/controllers/tournaments_controller.rb
app/helpers/pages_helper.rb
app/helpers/tournaments_helper.rb
app/models/user.rb
app/views/devise/confirmations/new.html.erb

Obr. 2


Pull requests
#1 OPEN
feature/registration
master
Test review
Overview
Commits
Activity
Author
neonosos
Reviewers
Description
Test pullrequestu ;)
Comments (1)
lubosch
Such cool so much wow so lines
Reply • Create task • 3 hours ago
#555 ; BUG ; Incorrect email validation
The input field #user_email is not correctly validated (aa.bbb@sk passes the validation)
Comment
Cancel

obr. 3


Tournament Manager
Search:
Tournament Manager
Overview
Activity
Roadmap
Backlogs
Task board
Releases
Issues
New issue
Charts
Gantt
Calendar
Repository
Settings
New issue
Tracker *
Task
Private
Subject *
Subject obeying methodic
Description
Description obeying methodic
Status *
To Do
Priority *
Normal
Assigned To
Parent task
Due date
Estimated time
Hours
% Done
0 %
Checklist
Files
Browse...
No files selected.
Watchers
Check
Matej
Igor
Peter
Jaroslav
Ľubomír
Marian
Martin
Create
Create and continue
Preview
www.ch-m.eu/scrum/projects/tournament-manager/issues/new

obr. 4