Metodika č. 2016 – 3

Verziovanie zdrojového kódu a integrácia

	VYPRACOVAL
FUNKCIA	Manažér verzií
MENO, TITUL	Helmut Posch, Bc.
DÁTUM	22/03/2016
PODPIS	

1. Účel a dedikácia metodiky

Účelom verziovania kódu je jednoduchšia kolaborácia pri jeho tvorbe. Každý programátor vytvára svoj vlastný kód (verziu) a všetky sa neskôr spoja do jedného funkčného celku. Taktiež je jednoduchý návrat k rôznym verziám čo zjednodušuje hľadanie chýb. Integrácia zahŕňa nasadenie nových verzií na produkčnú doménu.

Slovník pojmov

Bitbucket - úložisko pre repozitáre spravované verziovacími systémami napr. Git
Git - verziovací systém zdrojového kódu

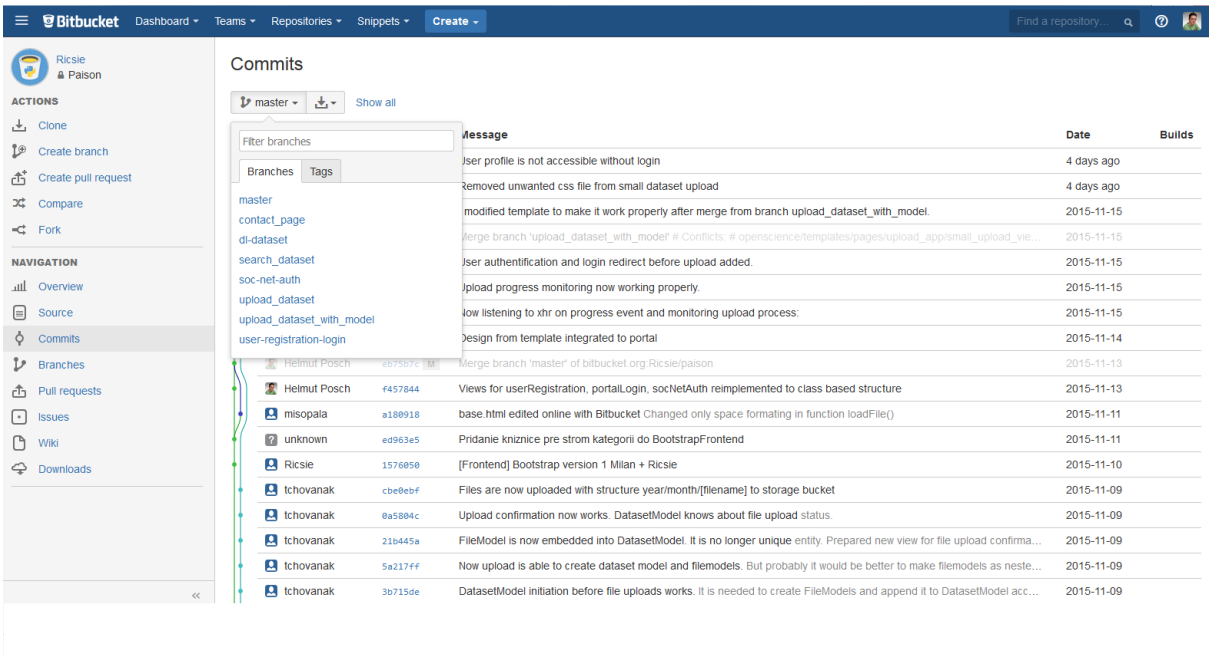
2. Inicializácia lokálneho úložiska

V projekte používame verziovanie kódu pomocou úložiska Bitbucket *(bitbucket.org*) a verziovací systém Git. K nasledujúcim častiam sa predpokladá, že programátor má pridelený prístup k projektu v aplikácii Bitbucket-u a nainštalovaný systém Git.

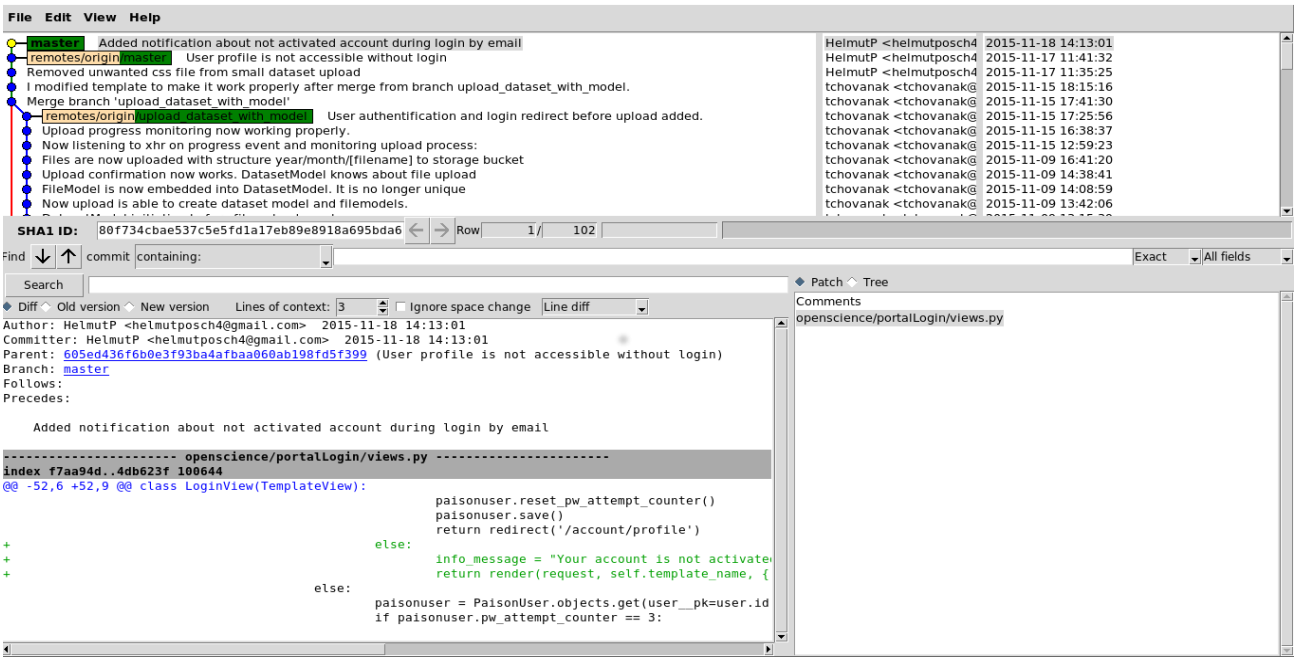
Aby bolo možné pridávať nový kód do projektu, je nutné si vytvoriť lokálne úložisko s projektom. Programátor si vytvorí vlastný priečinok pre projekt kdekoľvek vo svojom počítači. Po nastavení sa do vytvoreného priečinku spustí príkaz *git init.* Po tomto kroku si naklonuje projekt k sebe príkazom *git clone git@bitbucket.org:Ricsie/paison.git.* V tomto stave je lokálne úložisko úspešne inicializované.

3. Prehliadky verzií a zmien kódu projektu

Verzie vo vzdialenom repozitári je možné si prezerať pomocou aplikácie *bitbucket.org.* Je možné vybrať si jednotlivé vetvy. Verzie je možné pozerať ako zoznam zmien alebo ako celý zdrojový kód. Vetvy sú tiež zobrazené v samostatnej časti, podľa toho či už sú spojené s hlavnou vetvou alebo nie.



Zmeny v lokálnom úložisku je možné sledovať pomocou príkazu *gitk --all*, čo predstavuje grafické rozhranie s mapou verzií projektu. Pre zobrazenie lokálnych zmien v textovej forme sa používa príkaz *git status.*


File Edit View Help
master Added notification about not activated account during login by email
remotes/origin/master User profile is not accessible without login
Removed unwanted css file from small dataset upload
I modified template to make it work properly after merge from branch upload_dataset_with_model.
Merge branch 'upload_dataset_with_model'
remotes/origin/upload_dataset_with_model User authentification and login redirect before upload added.
Upload progress monitoring now working properly.
Now listening to xhr on progress event and monitoring upload process:
Files are now uploaded with structure year/month/[filename] to storage bucket
Upload confirmation now works. DatasetModel knows about file upload
FileModel is now embedded into DatasetModel. It is no longer unique
Now upload is able to create dataset model and filemodels.
HelmutP <helmutposch4 2015-11-18 14:13:01
HelmutP <helmutposch4 2015-11-17 11:41:32
HelmutP <helmutposch4 2015-11-17 11:35:25
tchovanak <tchovanak@ 2015-11-15 18:15:16
tchovanak <tchovanak@ 2015-11-15 17:41:30
tchovanak <tchovanak@ 2015-11-15 17:25:56
tchovanak <tchovanak@ 2015-11-15 16:38:37
tchovanak <tchovanak@ 2015-11-15 12:59:23
tchovanak <tchovanak@ 2015-11-09 16:41:20
tchovanak <tchovanak@ 2015-11-09 14:38:41
tchovanak <tchovanak@ 2015-11-09 14:08:59
tchovanak <tchovanak@ 2015-11-09 13:42:06
SHA1 ID: 80f734cbae537c5e5fd1a17eb89e8918a695bda6
Row 1 / 102
Find commit containing:
Exact All fields
Search
Diff Old version New version Lines of context: 3 Ignore space change Line diff
Patch Tree
Comments
openscience/portalLogin/views.py
Author: HelmutP <helmutposch4@gmail.com> 2015-11-18 14:13:01
Committer: HelmutP <helmutposch4@gmail.com> 2015-11-18 14:13:01
Parent: 605ed436f6b0e3f93ba4afbaa060ab198fd5f399 (User profile is not accessible without login)
Branch: master
Follows:
Precedes:
Added notification about not activated account during login by email
---------- openscience/portalLogin/views.py ----------
index f7aa94d..4db623f 100644
@@ -52,6 +52,9 @@ class LoginView(TemplateView):
paisonuser.reset_pw_attempt_counter()
paisonuser.save()
return redirect('/account/profile')
+ else:
+ info_message = "Your account is not activate
+ return render(request, self.template_name, {
else:
paisonuser = PaisonUser.objects.get(user__pk=user.id
if paisonuser.pw_attempt_counter == 3:

4. Stiahnutie aktuálnej verzie projektu

Prvým krokom k možnosti zmeniť kód projektu je stiahnutie si jeho aktuálnej verzie. Každý programátor je povinný spraviť tento krok. Vykoná tak príkazom *git pull,* čím stiahne všetky vetvy a verzie projektu zo vzdialeného repozitára do lokálneho úložiska vo svojom počítači.

5. Hierarchia vetiev

Hlavnou vetvou v repozitári je vetva dev-master. Od tejto vetvy sa odvodzujú ďalšie vetvy a implementuje sa do nej nová funkcionalita. Pokiaľ je verzia systému na hlavnej vetve stabilná, manažér verzií ju spojí s vetvou master. Vetva master predstavuje verzie systému, ktoré sú nasadené na produkčnej doméne. Rozhodli sme sa pre takéto delenie vetiev aby sme mohli sledovať, čo a kedy bolo nasadené na produkciu. Vetva dev-master taktiež obsahuje konfiguráciu na vývojové prostredie a vetva master pre produkčné prostredie.

6. Pridávanie novej verzie

Postup pridávania nového kódu sa delí na dva spôsoby.

Prvým prípadom je, ak po pridaní nového kódu do projektu ostane projekt alebo vyvíjaná časť nefunkčná. Nefunkčnosťou chápeme zobrazovanie chybových hlášok alebo funkčnosť, ktorú nie je možné považovať za prezentácie schopný prototyp zákazníkovi / používateľovi. V tomto prípade je programátor povinný si vytvoriť novú vetvu (branch)(pozri nižšie.)

Druhým prípadom je ak programátor vkladá do projektu funkcionalitu, ktorá nebude mať zásadný vplyv na funkčnosť projektu. Ide predovšetkým o zmeny textového obsahu, jednoduchá zmena designu, jednoduché opravy chýb. V tomto prípade je možné vkladať kód (pozri nižšie) do hlavnej vetvy (dev-master).

a. Pridanie novej verzie (commit-u)

Pred samotným pridaním prebieha výber súborov, ktoré do novej verzie patria. Po spustení príkazu *git status* sa červenou farbou zobrazia súbory, ktorých zmeny nie sú zahrnuté do novej verzie a zelenou farbou súbory, ktoré sú zahrnuté do novej verzie. Súbor do verzie pridáme príkazom *git add <nazov subora>* alebo *git rm <nazov subora>* podľa toho, či bol v novej verzii súbor pridávaný resp. modifikovaný alebo zmazaný. Potvrdenie novej verzie vykonáme príkazom git commit -m <sprava>. Správa sa píše v anglickom jazyku a má nasledovnú štruktúru:

[<typ zmeny>] <popis zmien>

kde typ zmeny je jedna z možností v nasledujúcej tabuľke.

Typ zmeny	Kedy použiť
FIX	Jednoduchá oprava chyby (zmena malého počtu riadkov, bez ovplyvnenia iných častí projektu)
FEATURE	Vytvorenie novej funkcionality, ktorá rozširuje tú v predchádzajúcej verzii
ISSUE	Zásadnejšia oprava alebo reimplementácia existujúcej funkcionality
MERGE	Zmeny k spojeniu viacerých verzií
REFACTORING	Transformácia kódu do inej podoby bez zmeny funkcionality a obmedzenia funkčnosti

Každá verzia by mala byť atomická a nie je žiadúce viac funkcionalít vkladať do jednej novej verzie. Predpokladá sa, že verzia sa pridáva práve do tej vetvy, v ktorej je práve programátor nastavený.

b. Vytvorenie novej vetvy (branch-u)

Rozhodnutie, či vytváraný kód potrebuje novú vetvu by malo prebehnúť ešte pred písaním kódu. Ak chceme vytvoriť novú vetvu, nastavíme sa do verzie projektu, z ktorej chceme vetvu vytvoriť. Do vybranej verzie projektu sa nastavíme príkazom *git checkout <hash verzie>.* Novú verziu následovne vytvoríme príkazom *git checkout -b <nazov novej vetvy>.* Názov vetvy je v anglickom jazyku, slová sú oddelené pomlčkami a názov obsahuje najviac 5 slov, pričom prvé slovo je predpona dev (pokiaľ sa vetva odvodzuje od inej vetvy s predponou dev). Ďalšie slová majú popisovať zmysel vetvy.

7. Spojenie novej vetvy s hlavnou vetvou

Túto činnosť vykonáva programátor novej vetvy spolu s ostatnými programátormi, s ktorých veziami nastáva konflikt. Programátor novej verzie sa nastaví do hlavnej vetvy *git checkout dev-master.* A vykoná príkaz *git merge <nazov novej vetvy>*. Pokiaľ nenastane konflikt, Git vykoná spojenie sám a taktiež programátor môže vykonať celý proces sám. V prípade konfliktu je potrebné kontaktovať autora kódu, v ktorom je konflikt a dohodnúť sa na spoločnom riešení. Do hlavnej vetvy sa spájajú iba vetvy, ktoré sú plne funkčné. Tejto akcii predchádza prehliadka kódu popísaná v metodike Prehliadky zdrojového kódu.

8. Odovzdanie novej verzie do vzdialeného úložiska

Po vykonaní zmien a potvrdení novej verzie je možné odoslať lokálny stav repozitára do vzdialeného úložiska. Jednotlivé vetvy je možné odosielať príkazom *git push origin <nazov vetvy>*, prípadne odoslanie všetkých vetiev a zmien príkazom *git push*.

9. Integrácia nových verzií

Integrácia nových verzií na produkčnú doménu začína tým, že manažér verzií spojí vetvu dev-master s vetvou master. Následne zdrojový kód vetvy master vloží na produkčnú doménu/hosting. Nikto iný nemanipuluje s produkčným prostredím. Úlohou manažéra verzií je aj spustenie a kontrola všetkých databázových migrácií, ktoré sú potrebné na správne fungovanie novej funkcionality. Migrácie spúšťa v administrátorskom prostredí portálu.

Všetky vývojové vetvy sú spúštané na vývojovej doméne (dev-openscience.appspot.com). Každý z členov tímu má pridelenú vlastnú poddoménu vývojovej domény. Produkčná doména má adresu openscience-paison.appspot.com a je iba jedna.