**گزارش بخش پلاگین git در پروژه**

**هدف از این بخش از پروژه و معماری استفاده از آن:**

در این بخش از پروژه هدف ایجاد بستر یا راهکاری برای ارتباط با مخازن git و امکان استفاده از امکانات اصلی ارایه شده توسط git است. علت انتخاب git بعلت رایج و پیشرو بودن آن در بین برنامه نویسان که مخاطب اصلی پروژه هستند، مبیاشد. از جمله مواردی که در این پروژه برای پزوژه حایز اهمیت است، بشرح زیر است:

1. Pull یا قابلیت دریافت از یک مخزن دیگر
2. Push یا قابلیت ارسال یک commit به یک مخزن خارجی
3. Merge یا قابلیت دریافت از یک شاخه(branch) متفاوت و ادغام آن با شاخه جاری
4. Commit یا ایجاد یک snapshot از وضعیت کنونی فایل ها
5. Add/Remove کردن فایل به stage workspace
6. Clone یا بارگزاری اولیه کل یک مخزن
7. Initial یا ایجاد یک مخزن محلی(local)

نکته ی مهم تمام این موارد نحوه استفاده آن ها در پروژه می باشد. ما بدین منظور نیازمند راهکاری برای ارتباط low level و بدون استفاده از ابزار های اجرایی ارایه شده توسط خود تیم git است با مخازن git میباشیم. بعنوان مثال استفاده از یک API استاندارد ارایه شده توسط تیم git میتواند راهکار مناسبی باشد.

همچنین نکته ی مهم دیگر یکپارچگی تیم و استفاده از زبان Ruby برای انجام این بخش از پروژه میباشد.

**کتابخانه مورد استفاده و مشکلات و نحوه پیاده سازی:**

در نتیجه در گام اول برای پیدا کردن پروتکل های ممکن برای ارتباط با مخازن git به جستجو پرداختیم. در واقع ما در این بخش انتخاب های محدودی داشتیم که خوشبختانه کتابخانه ای دقیقا با منظوری مشابه هدف این بخش از پروژه پیدا شد. این کتاب متن-باز ، با نام ruby-git-lib توسط یکی از برنامه نویسان تیم git و همچنین تیم ruby on rails در حال توسعه است و براحتی قابل دسترس است.

برای استفاده از این کتابخانه ابتدا باید آن را با دستورات ruby gem و نام ruby-git به کتابخانه های استاندارد ruby اضافه کنید. از این پس میتوان با دستور زیر از امکانات این کتابخانه استفاده کرد.

require 'git'

Git.whatever

در این بخش ما میبایست مجوعه ای از امکانات لازم برای پیاده سازی که از git انتظار میرفت، پیاده سازی میشد. بدلیل ضعیف بودن مستندات این کتابخانه که تنها در قالب یک Class Diagram بدون هیچ گونه توضیحی در آدرس زیر ارایه شده است، کار با این کتابخانه را بسیار دشوار میکرد.

http://rubydoc.info/github/schacon/ruby-git/Git

همچنین تازگی زبان ruby مشکلات کار را دوچندان مینمود. نکته مهمی که در پیاده سازی توابع کار با git باید ملاحظه میگردید، نحوه ارتباط آن با سایر بخش های پروژه و معماری Plugin-base بودن آن، که از ویژگی های اصلی پروژه می باشد، بود. در نتیجه با همفکری که در جلسات تیمی شکل گرفت، قرار بر استفاده تک-تابعی از امکانات لازم برای کار با git شد. بدین معنا که برای مثال هنگامی که تابعی برای پیاده سازی دستورات مختلف git ( مثلا add file) نوشته شد، آن تابع در قالب یک فایل ruby(\*.rb) و با نام عملکرد آن به معماری decentralize بخش کنترلی پروژه ارایه بگردد و بخش کنترلی با کمک یک فایل پیکربندی با گرامر قراردادی تیم، از آن استفاده کند. نمونه ای از این فایل پیکر بندی که مربوط به بخش clone میشود، در زیر آمده است:

<!DOCTYPE config>

<config>

<execute>ruby</execute>

<menu file="config/git/clone.rb"

name="clone"

icon = "clone\_pic.png">

<arg>PATH</arg>

<arg>PROJECTNAME</arg>

<arg>URL</arg>

</menu>

این فایل پیکر بندی امکان استفاده از منو های داینامیک را برای بخش GUI نیز بسادگی فراهم میکند.

این نحوه plugin بندی با کمک یکی از امکانات کتابخانه Qt که امکان اجرای قطعه کدی از یک زبان دیگر درخود هسته برنامه را میدهد، پیاده سازی و به مرحله اجرا رسیده است. همانطور که مشاهده میشود، در بخش پایانی clone نوع پارامتر های ارسال شده و مورد نیاز تابع clone.rb نیز مشخص شده است. این پارامتر های ارسالی در فایل ruby بطریق دستگیره(Handler) ARGV قابل استفاده میباشند:

workingDir = ARGV.shift

projectName = ARGV.shift

url = ARGV.shift

Dir.chdir workingDir

g = Git.init

g = Git.clone(url, projectName)

از جمله مشکلات اصلی بعد از پیاده سازی توابع اصلی، پیاده سازی توابع مرتبط با بخش information gathering کار بود. هدف از توابع این بخش بدست آوردن اطلاعات کافی برای نمایش نمودار های پروژه بود. اطلاعات مورد نیاز به دو دسته اصلی تقسیم میشدند:

1. تعداد commit های هر فرد در یک مخزن
2. تعداد commit های هر فرد بر روی هر فایل که نمایانگر اهمیت فایل ها نسبت به هم میگردد

بعلت عدم وجود اطلاعات raw در رابطه با بالا به ناچار میبایست این اطلاعات را با مشاهده اطلاعت دقیق هر commit و فایل های آن بدست آورد. همچنین بعلت نحوه ذخیرسازی git برای مورد دوم، نیاز به مقایسه اطلاعاتی هر فایل با نسخه پیشین همان فایل در commit پیشین آن شدیم که برای این قسمت بشکل زیر، نتایج را در یک hash میتوان بدست آورد:

@g.log.each do |l|

if first

#puts "First"

first=false

lc2=l

else

lc1=lc2

lc2=l

#puts @g.gcommit(lc1).author.name

if c[@g.gcommit(lc1).author.name]==nil

c[@g.gcommit(lc1).author.name]={}

end

changedFileArray = @g.diff(lc1,lc2).stats[:files].keys

for f in changedFileArray do

if c[@g.gcommit(lc1).author.name][f]==nil

c[@g.gcommit(lc1).author.name][f]=1

else

c[@g.gcommit(lc1).author.name][f] = c[@g.gcommit(lc1).author.name][f] + 1

end

end

end

#puts l1," ",l2

#puts @g.gcommit(l1).gtree.blobs.keys

#puts @g.gcommit(l2).gtree.blobs.keys

end

از دیگر مشکلات این کتابخانه عدم وجود یک exception handler مناسب برای خطاهای ممکن بود که کار را برای جلوگیری از بروز خطا سخت میکند و در این پروژه سیاست کنترل خطا در سمت front-end اتخاذ شد.