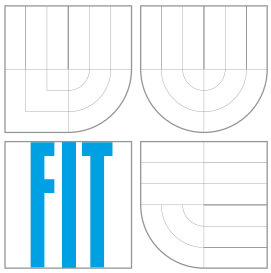
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ANALYSIS AND NOTIFICATION OF NEW RESULTCLOUD SUBMISSIONS

ANALÝZA A OZNÁMENÍ O NOVÝCH RESULTCLOUD VÝSLEDCÍCH

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE

BOHDAN IAKYMETS

SUPERVISOR
VEDOUCÍ PRÁCE

Mgr. Bc. HANA PLUHÁČKOVÁ

BRNO 2016

Abstract

Assignment of this bachelor work is design and implement mechanism for analyzing and notifing user about interesting changes in test results.

Abstrakt

Zadaní teto bakalařské práce je navrhnout a implementovat mechanism pro analyzu a oznamení uživatele o zajimavých změnách v vysledcích sady testů. Mechanismy museji byt snadno rozširitelné a dobře integrovane v ResultCloud.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Reference

IAKYMETS, Bohdan. *Analysis and Notification of New ResultCloud Submissions.* Brno, 2016. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Pluháčková Hana.

Analysis and Notification of New ResultCloud Submissions

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bohdan Iakymets
May 13, 2016

Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

Contents

Chapter 1

Introduction

The main goal of the project is to design and develop a mechanism for analyzing and notifying users about interesting changes in new uploaded submissions. The mechanism must support a few types of notifications and has possibility to extend types. The changes must also have interface for presentation results. Interested people would be able to set notifications for interesting results of the analyzes.

It is important and useful because the biggest part of the test results are not interesting and has useless information, like the same testing results. Thus, the main goal of the analyzing is to find the interesting results and show them to user.
Analyzing of submissions is very important because a lot of results are useless, in most cases they are the same data, so it doesn't give any important information. Analyzing helps to save developer's time, it finds useful information and notifies developers or other interested people about that and thus anyone at any time can easily find needed information, or to see statistics of project.
Firstly, I must learn inner architecture of ResultCloud. How it works. This help me to use better all the opportunities in design and programming that mechanism.

Chapter 2

Introduction in ResultCloud

This chapter is describe how ResultCloud work and all important moments for this bachelor's work, like what is submissions and how import of new test cases work.

2.1 What is ResultCloud

ResultCloud is a system for management of long-term testing results. This means that ResultCloud collect testing results of some project and compare it, so developer can easily find the difference between them. Currently, there don't exist new, modern instruments for the collect and the presentate testing results in readable form. As written by Fillip Matys: "Tools which solve that problem [3], are too old and fall behind all modern applications. One of the biggest problem of that tools is no opportunity to expand and with growing market of different mobile devices with internet connection not able to present data in responsive form." So all instruments that we have now is too old, and doesn't extendable. But in ResultCloud parsing and management doing by modules. Each module is written for one type of testing results. For example module "DejaGnu summary v1.0" can parse and show only SystemTap results.

ResultCloud is a complex system. ResultCloud consist of two parts frontend and backend. Frontend is a part on the client side, compited with using of AngularJS. Angular in asynchronous format connecting with backend part. Backend also dividing on the smaller part. On the top of hierarchy are Controllers. This hierarchy can be seen in image**??**. Angular connect directly with controllers. Controllers has only one mission, to get request and parsed data and give it to Services. Services are very important part, they get data and use different database entities, other services, plugins for returning result to the controller, which return it to client.

For connecting with database there is exist a Driver that connected to database,

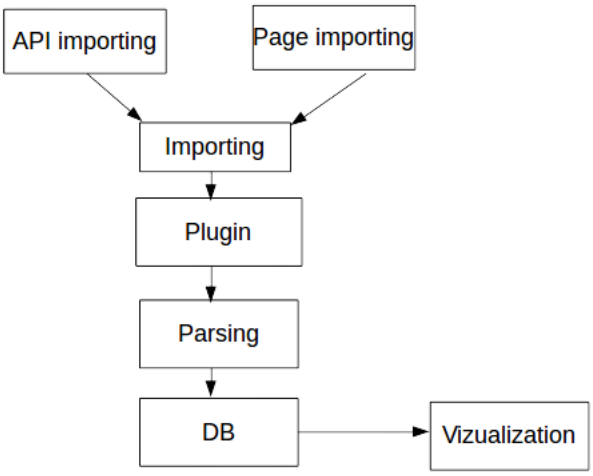


Figure 2.1: ResultCloud architecture

and use data access object (DAO). DAO consists all basic methods for working with database. Every entity must have DAO, which inherit base DAO, and then system will work with entity through entity's DAO. Results of SELECT query converting to TSE (Test-Suite-Entity) object. TSE object helps to work easier with entities.

Data organization in ResultCloud is represent in Plugins. Plugin is a system for parse and save test's results in specific format. In the past plugins has all demanded for self-installation, parsing and vizualization data, now vizualization is shared. Each plugin has own implementation of Parser to parse input data, it is saved in a Project, Submission, etc. Hierarchy of shared entities is following: Projects contains Submissions, Submissions is a results of a single series of tests, which also divide to Categories, Categories are divided to TestCases, and TestCases to Results.

In this bachelor's work I will analyse Submissions, their Results, and other stuff that may be interesting for people.

2.2 What is Submission

As I wrote behind, the submission is a single testing result. The smallest part of every submission is Result. This part contains result of a single test from series of tests. All other part like Categories, TestCases are only organization unit.

There are two ways to import a new Submission: first is from a web page, second is using an API. Then include plugin for parse submission file. Every plugin has a class Parser, for parsing files and putting them into Database (DB). When client send file to `ImportController` or to `import` class, it call `ImportService`, which find demanded plugin in DB, then include plugin's class `Parser` and call method *ParseImport*. Parser returns to `ImportService` `SubmissionTSE` object, which consists all parsed data as TSE objects. `ImportService` than save it to DB and return successful result to Controller or API class.

Chapter 3

Analyzer Mechanism Design

All analyzers must somehow unite into one working system. There must be mechanism for that. Mechanism must be not complicated and easy to extending. Thus it must easy to controll all analyzers and work with their results. In this chapter is described proposals about how better impelement analyzer mechanism.

3.1 Architecture

There two type of architecture: module and built-in. Module type mean that Mechanism would be divided to the modules, like "Divide and Conquer", one of the main advantage is easy extending. Second type is built-in, which means mechanism would be built-in whole ResultCloud system, one of the main advantage of this type is working speed.

I prefer first method, because difference in speed beetwen them would be to small, but easy extedning advantage is that what mechanism need. So lets start from main part, kernel of whole mechanism, `AnalyzerController`.

3.2 AnalyzerController

`AnalyzerController` would get all existing analyzers and use it. Mechanism also would provide entity for saving analyze data. Analyzer can't work with DB because in practice it is usually to divide work between separated modules, like "Divide and Conquer", so analyzer should only analyze input data and visualize it, that all. Centralized method good for that case because user don't need to load needed analyzers and work with DB, all this stuff do `AnalyzeController`.

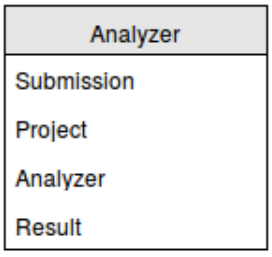


Figure 3.1: Analyzer entity

3.3 Analyzer entity

In the picture presents entity Analyzer **??**, which contain four attributes. Attribute *Submission* would have ID of the submission, that analyzer results belongs to. *Project* is

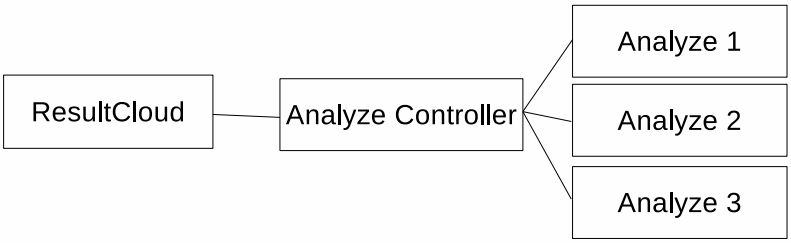


Figure 3.2: Analyzer structure

alternative attribute to *Submission*, it would have ID of the project. *Analyzer* contain machine analyzer ID. And *Result* contain analyzer results, it is text attribute, every analyzers have own output results format.

3.4 AnalyzerController structure

As can be seen in the image **??**, `AnalyzerController` is center part of whole analyzer mechanism. When application start `AnalyzerController` find all available analyzers, it is good for optimization. In `AnalyzerController` would be implement easy analyzer controlling. One method must run all analyzers which support current submission's plugin, and return result which `AnalyzerController` write to DB. Methods for vizualizating data.

3.5 Analyzers

This section describe design of Analyzers by itself and how it would be connected with analyze controller.

3.5.1 Analyzer design

For right connecting with analyze controller, analyzer must have first of all static constant attribute with unique machine ID (under it ID, analyzer would be identified in Analyze entity), method for getting and processing data (name of the method must be the same in all Analyzer classes) and two functions for vizulizating data (Visualize, VisualizeSingle). Vizualizating functions will get data from AnalyzerController which get it from DB and return it in JSON.

Method for processing data get in parameters: array of submission, new submission and plugin name. It must return ValidationResult object, with string in Data attribute, or array of string if it has a few resuls, or it can return empty result, with null in Data attribute.

Also analyzer have attribute called *is_interesting* which contain status of previous analyzing, and if analyzing results will be interesting it return true, in other case false, it need to notifying about only interesting submissions.

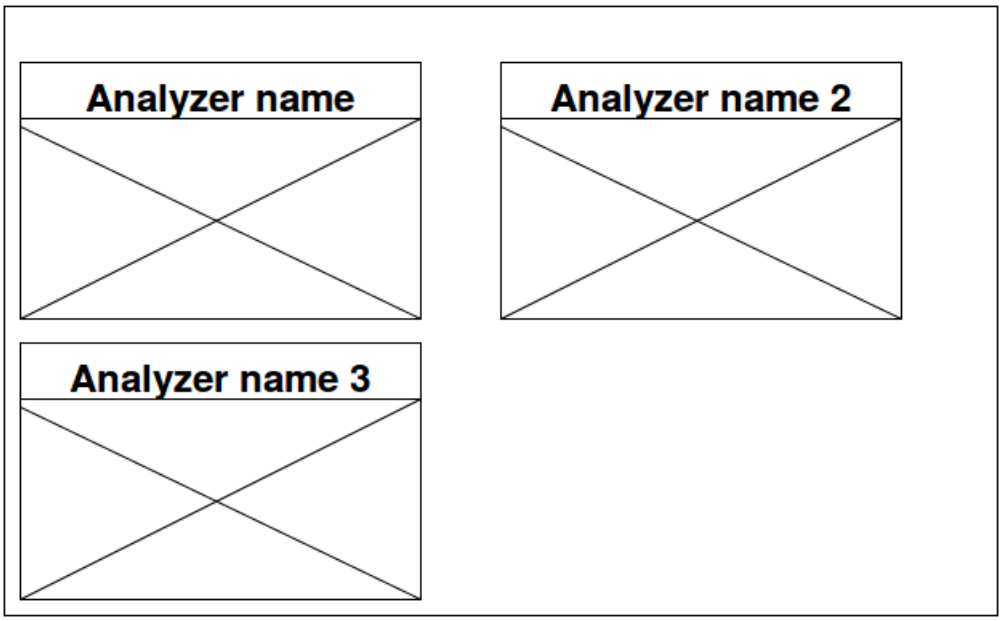


Figure 3.3: Template

3.5.2 Analyzer vizualisation

Every submission has own analyzing results, that results will be presenting in individual page. How can be seen in image**??** every analyzer have some space on that page. Because different analyzers use different methods for vizualization there is needed to specificate different possibilites to vizualize information. As mentioned in first chapter for vizualizating data ResultCloud use AngularJS and templates, so every analyzer must have own template and AngularJS directive. On the image can be seen analyzers results page layout.

3.5.3 Kinds of Analyzer

Next step is propose some kind's of analyzers. Here is analyzers which results would be interesting for programmers.

- *Find strange changes like if result has a long time the same value and than it change*, it would be interesting because a lot of test cases has long time same result, so most of time it is just useless information, but changing is interesting and useful for developer.
- *Check a changes in tests, like if some test which is contained in all previous submissions just dissapear*, it would be interesting and useful because it is changing in test cases by itself.
- *Check if some test had a long sequence of some bad value like FAIL or ERROR and then take a PASS, but after take FAIL or ERROR again*, it would be interesing because by using this information developer can find why test always failing.
- *Check changes from UNTESTED to some result*, it would be interesting because unused test case started to be in use.
- *Check if presented a new tests*, it would be interesing because new tests can bring new useful information.

- *Check GOOD, BAD, STRANGE changes in tests*, it would be interesting because all changes can bring new useful information about programe work.
- *Check if count of bad results is get maximum*, it would be interesting because it notify developer about that changes caused a lot of bad results.

Chapter 4

Analyzer Mechanism Implementation

In this chapter is described mechanism implementation. Mechanism implemented in PHP and JS, becaue that languages was used for implementation ResultCloud.

4.1 Structure

Because analyzers are not a plugins or any else components in ResultCloud, analyzers will be *extentions*. Whole system have own directory *analyzing*. Which contain one directory for analyzers - *analyzers*, and one for templates - *templates*. Root directory also would contain `AnalyzeController`.

Analyzing starts only when new submission would be inserted into DB, in `ImportService` class.

4.2 Entity

Analyzer entity was converted into ResultCloud acceptable format. As a result was created three classes: `AnalyzerDao`, `AnalyzerTSE`, `AnalyzerService` and edited table installation class. `AnalyzerDao` class for working with *Analyzer* table. `AnalyzerTSE` class for easy working with `AnalyzerDao` returned data. `AnalyzerService` class for different more complicated operations with data.

4.3 Analyzing

4.3.1 AnalyzeController

`AnalyzeController` is a kernel of whole analyze mechanism. `AnalyzeController` implemented like static class (but PHP does not support static classes, thus all methods are static), because create more than one class object unnecessarily. Image**??** good describe how whole mechanism is implemented. `AnalyzerController` connecting with analyzers, and through `AnalyzeService` write data to DB and get it from DB.

When `AnalyzeController` be included, it execute *InitAnalyzers* method, that scan *analyzers* folder, put all available analyzers together and save it to *$AnalyzerList* attribute. Method *GetAnalyzersList* will return `LINQ` object with *$AnalyzerList*.

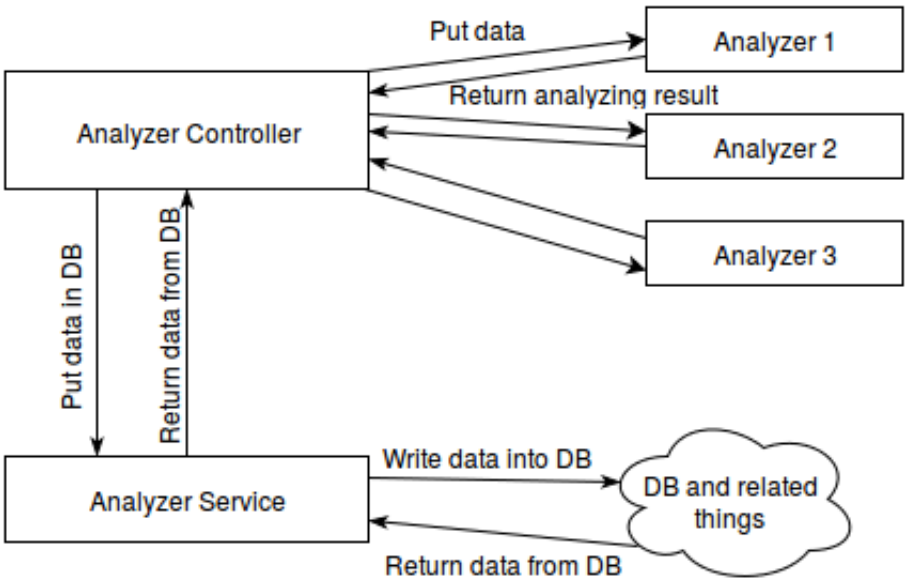


Figure 4.1: Analyzer implementation

analyze method get all analyzers from *$AnalyzerList*, and call it analyzer method. Than returned value, or values it write to DB, and check if results are interesting by getting boolean value from analyzer method *isInteresting*, if results are interesting it add analyzer ID to *$interesting_analyzers* array. Analyzers ID which has interesting value can get by method *GetInterestingAnalyzers*. Method *analyze* get like parameters: currently uploaded submission - *$submission*, `LINQ` object with older submissions - *$submissionList* and plugin name - *$plugin*. Returning `ValidationResult` object with the analyze status.

4.3.2 Analyzer1

`Analyzer1` is simple analyzer created like example of analyzer structure. Analyzer get new submission and the last one, and compare it, if it has differences, analyze results became interesting. There are three categories of changes: GOOD, BAD, STRANGE. Output result format is JSON. It support only ŝystemtapp̈lugin.

Every analyzers must have method *analyze*, which analyzing input submissions according to plugin name. Parameters are the same as has method *analyze* in `AnalyzeController`. Output results are in format JSON, it use JSON because of it simplicity. Attribute *$is_interesting* is boolean type, and became *true* only if analyzing results are interesting, otherwise it's false. As mentioned in previous part, analyzer has method *isInteresting*, which return value of *$is_interesting* attribute.

ANALYZER_ID is constant attribute, that contain unique analyzer ID, that ID is used in *Analyzer* table, like analyzer identifier. *JS_CONTROLLER* is also constant attribute which contain name of JavaScript file with AngularJS directive, it is used for vizualization analyzing results.

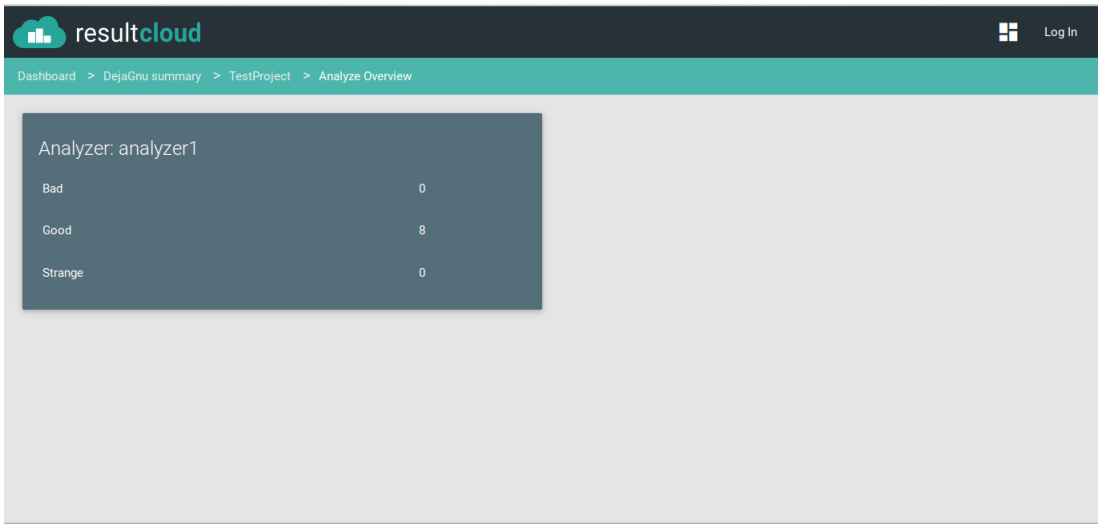


Figure 4.2: Implemented template

4.4 Vizualization

For vizualization data ResultCloud use AngularJS. AngularJS is JavaScript MVC (Model-View-Controller) framework, every page has own controller, thus analyzer page must have it too. `AnalyzeController.js` is file that contain controller for result page. *analyze.html* is a page template. Some of the page, that contain several sort of data, building with the simplest part *Components*, each component has individual settings, and individual Angular `directory`. Each component has `backend`, `frontend` folder and configuration file *config.xml* with all settings and supported plugins. Backend folder consists `CBuilder` class, which return prepared for presenting data.

Analyzer page would use only one component `analyzeOverview`.It wouldn't have any settings, and will support all plugins. Angular directory first of all get array of analyzing results for current submission, than for each analyzer find own Angular directory, which put analyzer data to template and present it. There some interesting part of code, how is implemented inserting analyzers directive into `analyzeOverview` component template:

```
$scope.buildAnalyzerView = function (key) {
\\ Check if key not empty
    if (!$(key).length) {
     \\ Make new scope clone from rootScope
        var data2 = $rootScope.$new();
        \\ Include into cloned scope analyzer data
        data2.data = $scope.data[key];
        \\ Compile analyzer directive tag with cloned scope
        var el = $compile('<' + key + '/>')(data2);
        \\ Put result into page
        $("#"+key).append(el);
    }
}
```

Image?? presenting how it actually look.

`CBuilder` class for `analyzeOverview` get `stdClass` object with attribute *Submission* - submission ID. And call `AnalyzeController` method *VisualizeBySubmission*. *VisualizeBySubmission* get submission ID, for each analyzer get last inserted result, and give it to analyzer's method *VisualizeSingle*, which parse results and return it like array. Then *VisualizeBySubmission* put vizualization data together into associative array the key analyzer ID and value analyze results, and return it.

Chapter 5

Notification design

Notification mechanism must be flexible, and easy extended. This chapter contain proposals for implementing notification mechanism.

5.1 Architecture

Like in case with Analyze Controller, I would divide notifications methods to the separated classes and Notification Controller will control them. But as opposed to analyzers there will be several types of notifiers. First type is public notifiers, it means notifications would be send into some shared or public resources, like *Twitter* for example. Private - means it notify each user separetly. According to this private notifiers must have settings, where user can check if he want to get notifications or not, and other options.

5.2 Notification Controller

Notification controller would have method for easy controlling notifications, which get all needed fields, like *title*, *body*, *bodyShort* (for resources that accept only small messages), *adreses* (list of all adreses that must recieve notification) and according to them send notifications. Also `NotificationController` must provide methods for getting private and public `Notifiers`. When it will be included, first of all it scan space for available `Notifiers`.

5.3 Notification settings

There must be mechanism for easy adding settings to `Notifier`, and settings must be present in user settings without editing any template for it. But user can editing only private settings, because public notifier settings are shared with other users, ResultCloud don't support user hierarchy, thus nobody can edit public notifier settings.

ResultCloud provide good tools for working with settings, like entities `TemplateSettings` and `TemplateSettingsItem`. `TemplateSettings` is for saving into DB setting template information, like setting type, setting name and etc. `TemplateSettingsItem` is for saving settings value.

5.4 Notifier

`Notifier` is a base part of notification mechanism, without at least one notifiers it is useless. `Notifier` provide notification by itself, each notifier has own notification method, for example, by email, or Facebook.

5.4.1 Notifiers architecture

Every `Notifier` for right work must have unique identifier(ID), by that ID `NotificationController` would identify notifiers, and in settings template it would have reference to notifier by it ID. `Notifier` must have one method for notification, and one method with settings. Each notifier has the same settings, thus that method can be pick out into some base paretn class, which would be extended by notifiers classes. `Notifier` must have attribute that identifies it like private or public notifier.

As a notify method in `NotificationController`, notify method in `Notifier` get the same parameters, except address, address will not be associative array, but simple array with addresses supported in that notifier.

5.4.2 Notifier types

Within the confines of that bachelour work, there must be implemented these types of notifiers:

- Email - notifies will be sent by email, this is private notifier
- Twitter - notifies will be sent into prepared twitter account, public notifier
- RSS - notifies will be present in RSS file, public notifier

Chapter 6

Notification mechanism implementation

This chapter contain description of notification mechanism implementation. As can be seen in image, `NotificationController` is a kernel, all notifiers are extended from `BaseNotifier` class.

6.1 Structure

Like analyzers, notification mechanism not a plugin or any else ResultCloud kernel part, thus it also be in *extentions* folder, in own *notification* folder. In root directory is located also `NotificationController`. All notifiers are located in *notifiers* folder.

Notification start only if analyzer return interesting result.

6.2 Settings

Basically all notifers will have same setting, this settings would enable or disable notification for notifier. But as mentioned in previous chapter, only private notifiers can use settings. Notification mechanism not working with settings, because list of addresses and notifiers must be assamble by those who sending, notification mechanism only get this list and send notification.

ResultCloud has tools for settings, not only in server side, but also in client side it has automatic form generation for settings. Here is example of default *getSettings* method in `BaseNotifier` class, for better understanding how set up settings:

```
public function getSettings()
{
    $settings = array();
    $settingsItem = array();

    \\ Setting label
    $settingsItem['label'] = "Get notifications by this way";
    \\ Setting ID for TemplateSettings entity
    $settingsItem['identifier'] = "get-notify";
    \\ Default value
```

```
    $settingsItem['default'] = "1";
    \\ Field type
    $settingsItem['type'] = TemplateSettingsItemType::CHECKBOX;
    \\ Is setting required
    $settingsItem['required'] = 'true';
    $settings[] = $settingsItem;

    return §$settings;
}
```

6.3 Notification

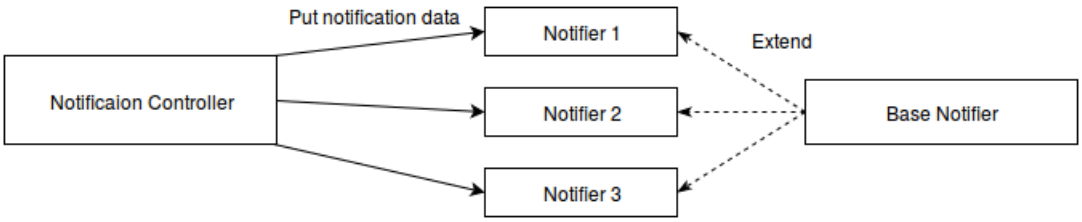


Figure 6.1: Notification mechanism implementation

6.3.1 NotificationController

Like in case with `AnalyzerController`, `NotificationController` is static to, all methods are static, because there is no reason to create more than one class instance in application. Notification mechanism scheme**??** showing how `NotificationController` connecting with other mechanism elements.

When `NotificationController` is included, first of all start method *preLoad*, that method scan *notifiers* folder, include and assamble array with all available notifiers. For notifing exist function *notify(title, body, bodyShort, to)*, where `title` - is title of notification message, `body` - longest body text, `bodyShort` - short body of the message not longer than 140 letters, `to` - is an array of all adresses with the key of notifier ID. Function *notify* calling notifier's function *notify*, only for notifiers which have their IDs in `to` parameter's key.

`NotificationController` also have different sorts of get methods: *getNotifyIds* (return IDs of all notifiers), *getPrivateNotifiers* (return array with IDs only for private notifiers) and *getNotifierById* (return notifier object by notifier ID).

6.3.2 Notifier

Every notifier must extend `BaseNotifier` class with default settings, and if needed define own method *getSettings* that in the begining call *parent* method. Also notifier must have unique ID in constant *NOTIFY_ID*. *NOTIFIER_PUBLIC* is constant, which contain boolean value, if notifier is public it contain *true*, otherwise *false*. The most important method is *notify*, it has same parameters like method *notify* in `NotificationController`, except last address parameter, notifier would get not associative array, but simple array with addresses.

6.3.3 Notify1

`Notify1` is a private notifier that send notifications by email. It get array with email addresses and via default PHP *mail* function send emails. Here notifier parameters:

```
const NOTIFY_ID = "notify1";
const NOTIFIER_PUBLIC = false;
```

6.3.4 Twitter

`Twitter` is a public notifier that send new twittes with some interesting information to twitter account. Now it connected to my account cyberbond95 . If analyzer would have interesting results it send new twitt to my account, and everyone can see it.

`Twitter` work with Twitter API by using `TwitterAPIExchange` library, which was suggested in Twitter API documentation. It has MIT license.

6.3.5 RSS

`RSS` is a public notifier that create or update *rss.xml* file in root folder. *rss.xml* present one news for each project, if `RSS` get news for already existing in *rss.xml* project, it update news, otherwise it just create new one.

`RSS` work with RSS by using `SimpleXML` that is default in mostly PHP versions.

Chapter 7

Conclusion

In this bachelor work was proposed mechanism of analyzing tests results and notifing users about it, mechanism must be easy extensible. There must be available more than one notifier for wider using.

Mechanism was built under ResultCloud system by using PHP and JS (AngularJS framework). It consists two part: Analyzers and Notifications. In analyzer part was implemented kernel (`AnalyzerController`) and one analyzer. In notification part was implemented kernel as well (`NotificationController`) and three notifiers. And was implemented other parts for integration mechanism into ResultCloud, like services for working with DB, different Angular directives for vizualization data.

Appendices

List of Appendices