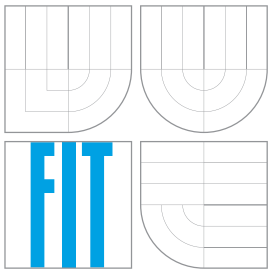
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



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

Tests results have mostly the same values, therefore they do not contain any important or interesting information. Developers must spend a lot of time for looking for something interesting in tests results, thus developer require tool for analysis results and in case finding interesting information notify user about it. This tool can save a lot of time. Assignment of this bachelor work is design and implement mechanism for analyzing and notifing user about interesting changes in test results. Part of the work is to learn ResultCloud and based on acquired knowledge to extend ResultCloud.

Abstrakt

Většinou výsledky testů jsou stejné a proto nenesou žádnou užitečnou informaci. Vývojáři musejí neustále probírat velké množství zbytečných informací, aby našli něco zajímavého. Tedy vývojář potřebuje nástroj pro analýzu testovacích výsledků a v případě nalezení zajimavé informace to oznámí uživateli. Tento nástroj ušetří spoustu času. Zadání této bakalářské práce je navrhnout a implementovat, mechanismus pro analýzu a oznámení uživateli o zajímavých změnách v výsledcích sady testů. Mechanismy musejí být snadno rozšiřitelné a dobře integrovatelné v ResultCloud. Součástí této práce je prostudování ResultCloud a na základě získaných znalostí rozšíření ResultCloud o analyzátor a oznamovatele. Nástroj je implementován pomocí AngularJS a PHP.

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 18, 2016

Acknowledgements

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

Contents

1 Introduction **3**

2 Introduction in ResultCloud **4**
2.1 Testing . . . 4
2.2 What is ResultCloud . . . 4
2.2.1 Internal structure . . . 5
2.2.2 Components . . . 6
2.2.3 Utilities . . . 6
2.2.4 Data store organization . . . 6
2.3 What is Submission . . . 7

3 Analyzer Mechanism Design **8**
3.1 Architecture . . . 8
3.2 AnalyzerController . . . 8
3.3 Analyzer entity . . . 8
3.4 AnalyzerController structure . . . 9
3.5 Analyzers . . . 9
3.5.1 Analyzer design . . . 9
3.5.2 Analyzer vizualisation . . . 10
3.5.3 Kinds of Analyzer . . . 10

4 Analyzer Mechanism Implementation **12**
4.1 Structure . . . 12
4.2 Entity . . . 12
4.3 Analyzing . . . 12
4.3.1 AnalyzeController . . . 12
4.3.2 Analyzer1 . . . 13
4.3.3 Analyzer2 . . . 14
4.4 Vizualization . . . 14
4.5 Evaluation . . . 15

5 Notification design **16**
5.1 Architecture . . . 16
5.2 Notification Controller . . . 16
5.3 Notification settings . . . 16
5.4 Notifier . . . 17
5.4.1 Notifiers architecture . . . 17
5.4.2 Notifier types . . . 17

6 Implementation of notification mechanism **18**
6.1 Structure . . . 18
6.2 Settings . . . 18
6.3 Notification . . . 19
6.3.1 NotificationController . . . 19
6.3.2 Notifier . . . 19
6.3.3 Notify1 . . . 20
6.3.4 Twitter . . . 20
6.3.5 RSS . . . 20
6.4 Evaluation . . . 20

7 Conclusion **22**

Bibliography **23**

Appendices **24**

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. Submission is a representation of results of tests series in ResultCloud. ResultCloud is a system for management of long-term testing results. The mechanism must support a few types of notifications (for example notifications by email or by twitter) and has possibility to add more. The analysis mechanism must also have interface for presentation results. An interested people would be able to get notifications about interesting results of the analysis.

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 users about that and thus anyone at any time can easily find needed information, or to see statistics of project.

In the first part of the work 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.

Next chapters describe (**Introduction in ResultCloud 2**) ResultCloud system how it works and why it useful for developers, what is submission in ResultCloud, (**Analyzer Mechanism Design 3**) analyzer mechanism proposal and (**Analyzer Mechanism Implementation 4**) implementation, (**Notification design 5**) notifications, why it is important, proposal and (**Notification mechanism implementation 6**) implementation notification mechanism and (**Conclusion 7**) conclusion about all done work.

Chapter 2

Introduction in ResultCloud

This chapter will describe what is software tests, how ResultCloud works, why it is useful and all important moments for this bachelor's work, like what is submissions and how importing of new series of tests results work.

2.1 Testing

Tests are one of the important part of software developing. Tests helps to developers find errors and fais in software. Many softwares has long developing proccess, this softwares demand an extensive series of tests. Series of tests is a file of test cases, it must test application or part of application.

Filip Matys in his bachelor's work use tool SystemTap for describe long-term tests. SystemTap has his own scripting language *stap*. SystemTap use for testing Linux kernel. SystemTap's series of tests is executed by tests interface DejaGnu. Series of tests can be consists more than one scripting files. SystemTap after running series of tests write output to text file.

2.2 What is ResultCloud

As I wrote in introduction, ResultCloud is a system for management of long-term testing results. This means that ResultCloud collects testing results of some project, build diagrams based on that results, compare it, so developer can comfortably look at results or easily find the difference between them. Currently, there do not exist new, modern instruments for the presentation of tested results in readable form, because all instruments that we have now is too old, and does not extendable, as written by Filip Matys in his bachelor's work: "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." [1]. ResultCloud has more advantage, for example in ResultCloud parsing and management doing by plugins. Plugin is a plug-in module, that can be connected to ResultCloud in any time. Each plugin is written for one type of testing results. For example plugin "DejaGnu summary v1.0" can parse and show only SystemTap results. Thus ResultCloud is extendable system.

ResultCloud is useful for developing applications because it provides tools for present, compare and work with long-term test's results. For developers it is quite hard to look

up for some information in a data bunch. But with ResultCloud developer only need to import results of tests series into ResultCloud, ResultCloud stores it and then presents it in a comfortable, readable form. For example: kernel of operation system need a lot of tests that collect into series of tests and for developer every time look for some interesting results take a lot of time, but ResultCloud store results in submissions, and then presents it like diagrams and lists of results, also provides some extendet tools for search interesting results, compares two or more submissions, thus developer can easily find or look at the results.

2.2.1 Internal structure

ResultCloud is a complex system. ResultCloud consists of two parts; frontend and backend. Frontend is a part on the client side, built with using of AngularJS. AngularJS is a JavaScript MVC (Model-View-Controller) framework which provides tools for building and working with web pages. In official documentation write next: „It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. Angular's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology“[2]. Angular asynchronous connecting with backend part.

Frontend (AngularJS)

For controlling whole page AngularJS uses controllers. Controller in Angular is defined by a JavaScript constructor function that used to increase Angular scope. Every controller has his own template, Angular automatically builds pages with template, according to data getting from controller's variable *scope*. Controller has variable *scope* that contains data for build page. AngularJS also has directives. Directives extend functionality of static HTML elements. A custom directive replaces the element for which it is actived by his own template. Thus it is easy to build complicated web pages that consists of more than one elements, and include external elements from other projects.

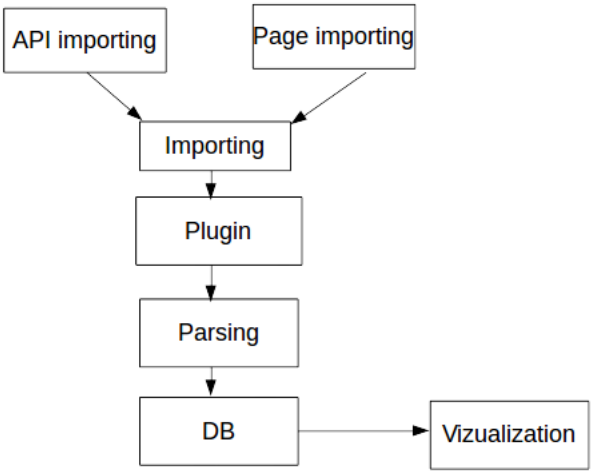


Figure 2.1: ResultCloud architecture

In ResultCloud, pages like login page, dashboard, project, plugin overview page and etc. use controller, but for building content using directives. Elements like submissions list, submission overview list and etc. using directives, it helps build more complicated pages, for example submission overview page using several directives, one for building list with results, one for building diagrams.

Backend

Backend is also divideed on the smaller part. Controllers are on the top of hierarchy. This hierarchy can be seen in image 2.1. Angular connect directly with controllers. Controllers are PHP classes. 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 returns it to client.

For connecting with database there is exists a `DatabaseDriver` that connected to database, and use data access object (DAO). DAO contains all basic methods for working with database. Every entity must have DAO, which inherit basic 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.

2.2.2 Components

All ResultCloud capabilities like submission overview, project overview and etc. are components, it means that they have their own settings and should be installed manually. Each component has his own configuration file in which defined component's ID, which plugins component support and other. Component system make possible to easily extend ResultCloud. But components can not save data or change it, components only prepare and vizualize data. Every component has his own frontend and backend part. Backend part is a file *CBuilder* with `CBuilder` class, method *Get* prepare data and return them. Frontend is a directive that get prepared data and vizualize them.

2.2.3 Utilities

ResultCloud has utilities, the most important utilities that was used in this bachelor's work, are described here:

LINQ – idea of this utility was gotten from C# and rewritten to PHP. `LINQ` is a class for simplest working with arrays, it has a lot of methods for getting array elements, filtering array elements and doing other operations with arrays that can not do default PHP functions. In application `LINQ` wrap standard array, thus make complicated operations more readable.

ValidationResult – this tool was created for data validation, it wrap validate object and enable to validate it. Many methods in ResultCloud using `ValidationResult`. `ValidationResult` has attribute *Data* which contain validate data, attribute *IsValid* which contain validation state and attribute *Errors* which contain array of strings that describe occurred errors.

2.2.4 Data store organization

Data organization in ResultCloud is represented in Plugins. Plugin parse and save specific formated test's results. 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, and save it in a Project, Submission, etc. Submissions of the same software are grouped to **Project**. In submission, data orginized to **categories**. If file with tests results has no categories, ResultCloud create „Default“ category. Categories are also divided to **TestCases**. TestCases contain **Results**.

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

2.3 What is Submission

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

There are two ways to import a new Submission: first is through the web page, second is using an API. For import in first way need to sign in like a user in ResultCloud and go to import page, fill all fields and press import button, then Angular send request with file to `ImportController`. To import in second way need sign in through ResultCloud API and send file with other parameters by POST request to `import` class.

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 contains 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 a mechanism for that. Mechanism must not be complicated and easy for extend. Thus it must easy to control all analyzers and work with their results. This chapter will describe proposals on how impelement analysis mechanism better.

3.1 Architecture

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

I choose first method, because difference in speed beetwen them would be to small, but easy extending advantage is that what mechanism demand. So let start from the main part, kernel of whole mechanism, `AnalyzerController`.

3.2 AnalyzerController

`AnalyzerController` would get all existed analyzers and use them. Mechanism also would provide entity for saving analysis data. Analyzer can't work with DB, because in practice it is normal to divide work between separated modules, thus analyzer only analyze input data and visualize it. Centralized method is good for that case because user don't need to load demanded analyzers and work with DB, all this operations do `AnalyzerController`.

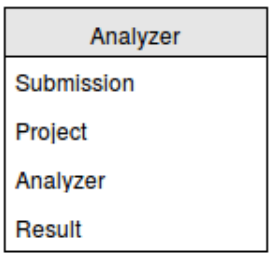


Figure 3.1: Analyzer entity

3.3 Analyzer entity

In the picture presents entity *Analyzer* 3.1, which contain four attributes. Attribute *Submission* has ID of the submission, that analyzer results belongs to. *Project* is alternative attribute to *Submission*, it has ID of the project. *Analyzer* contains machine analyzer ID.

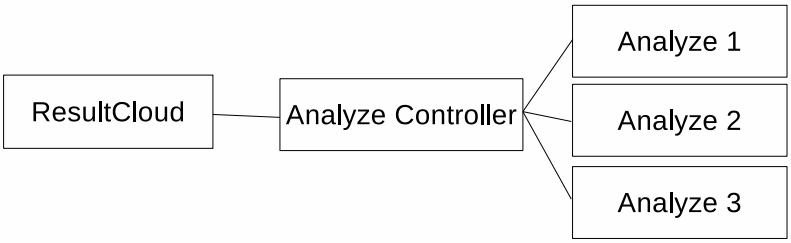


Figure 3.2: Analyzer structure

And *Result* contains analyzer results, this is a text attribute, every analyzer has his own output results format.

3.4 AnalyzerController structure

As can be seen in the image 3.2, `AnalyzerController` is center part of whole analyzer's mechanism. When application starts `AnalyzerController` finds all available analyzers, this is good for optimization. `AnalyzerController` is realize easy analyzer control. One method must run all analyzers that supports current submission's plugin, and returns result which `AnalyzerController` write to DB. Also `AnalyzerController` must consists methods for vizualizating data.

3.5 Analyzers

This section will describe design of Analyzers by itself and how it connected with analyzer controller.

3.5.1 Analyzer design

For correct connecting with analyzer controller, analyzer must has a 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 for all Analyzer classes) and two methods for vizulizating data (Visualize, VisualizeSingle). Vizualizating methods will get data from `AnalyzerController` which get it from DB and return it in JSON format.

Method for processing data gets in parameters: array of submissions, new submission and plugin name. It must returns `ValidationResult` object, with string in Data attribute, or array of strings if it has a few results, or it can returns empty result, with null in Data attribute.

Also analyzer has an attribute called *is_interesting* which contains status of previous analysis, and if analysis results is interesting it return true, in another case false, it need to notifying only about interesting submissions.

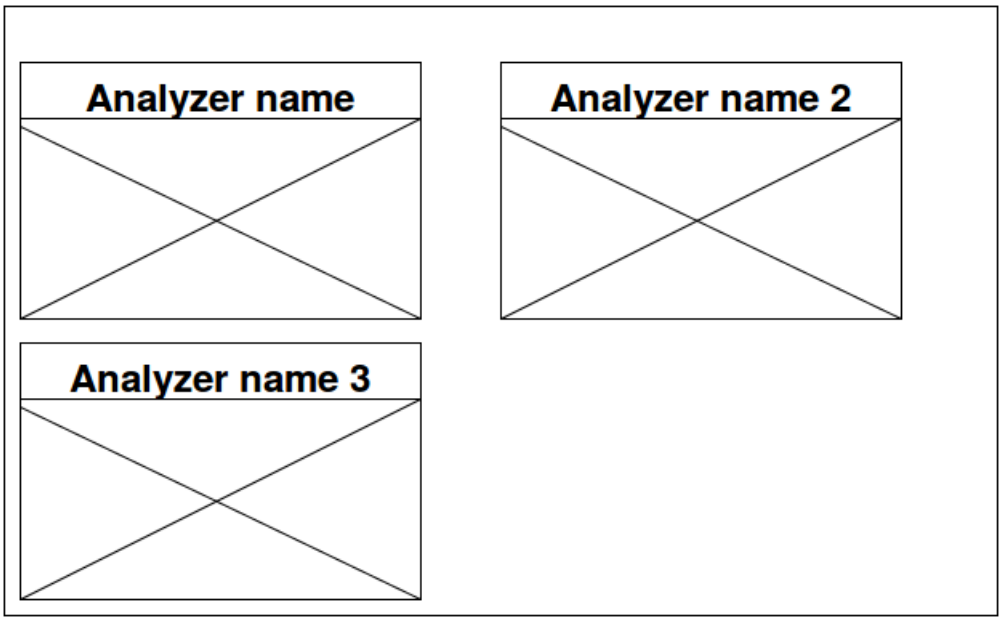


Figure 3.3: Template

3.5.2 Analyzer vizualisation

Every submission has his own analysis results, those results would be presented in personal page. Image 3.3 show that every analyzer has some space on that page. Because different analyzers use different methods for vizualization, there is need to specificate different possibilites to vizualize information. As mentioned in chapter 2 ResultCloud uses AngularJS for vizualizating data and templates, so every analyzer must have his own template and AngularJS directive. Image shows layout of analysis results.

3.5.3 Kinds of Analyzer

Next step is propose some kind's of analyzers. Here are analyzers whose 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 dissapear*, it would be interesting and useful because changes 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 using this information can help developer to 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 program 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

This chapter will describe mechanism's implementation. Mechanism implemented in PHP and JavaScript, because that languages was used for implementation ResultCloud. Mechanism would be implemented according proposals in previous chapter.

4.1 Structure

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

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

4.2 Entity

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

4.3 Analyzing

4.3.1 AnalyzeController

`AnalyzeController` is a kernel of whole analyze mechanism. `AnalyzeController` is implemented like static class (but PHP does not support static classes, thus all methods are static), because creates more than one class object unnecessarily. Image 4.1 describe good how whole mechanism is implemented. `AnalyzerController` connecting with analyzers, and through `AnalyzeService` writes data to DB and get it from DB.

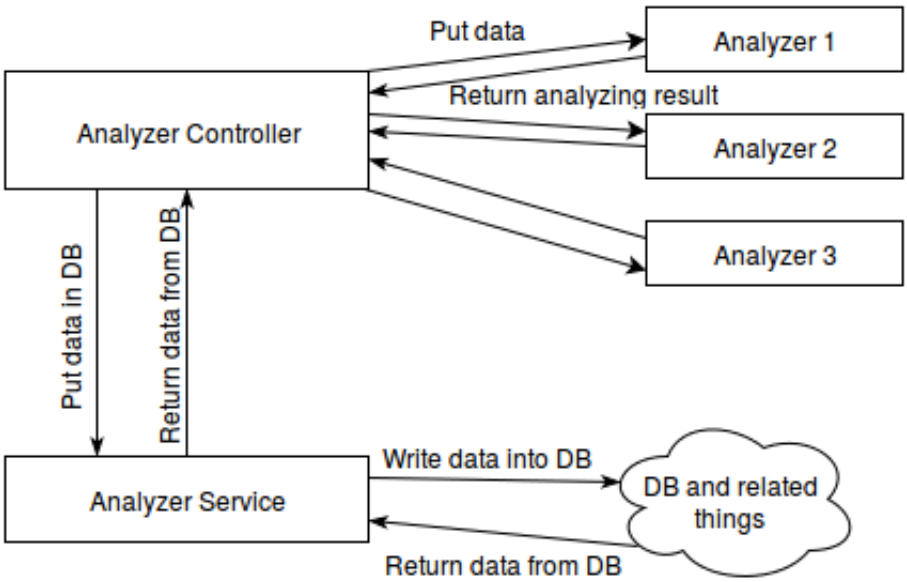


Figure 4.1: Analyzer implementation

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

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

4.3.2 Analyzer1

`Analyzer1` is a simple analyzer created like example of analyzer structure. Analyzer gets new submission and the last one, compare them, if they have differences, analyze results become interesting. There are three categories of changes: GOOD, BAD, STRANGE. Output result format is JSON. It supports only „systemtap“ plugin.

Each analyzer must has 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 uses JSON because of it simplicity. Attribute *$is_interesting* is boolean type, and became *true* only if analysis results are interesting, otherwise it is false. As mentioned in previous part, analyzer has method *isInteresting*, which return value of *$is_interesting* attribute.

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

`Analyzer1` get last imported submission and new imported submission, than by using *foreach* construction get each category from last imported submission (let call it *category1*)

and try to find category with the same name in new imported submission (let call it *category2*), if the same category not exists it get next category, othewise it doing the same with test cases, it get each test case from *category1* and try to find test case with the same name in *category2*, if test cases with the same names was found, it compare their results, if results with the same key has different value and differnce is GOOD (FAIL → PASS), BAD (PASS → FAIL), STRANGE (FAIL → ERROR), it increment variable that responsible for one of the differnce types and in the end return `ValidationResult` object with result in JSON format. Example of result in JSON format:

```
{
    "Good": 8,
    "Bad": 3,
    "Strange": 0
}
```

4.3.3 Analyzer2

`Analyzer2` is a sample analyzer, it looks for changes in submission's results, from UNTESTED to any other value. It has *ANALYZER_ID* - „analyzer2“ and *JS_CONTROLLER* - „analyzer2.js“. It support only „systemtap“ plugin. Output result's format is JSON. It works in the same way like `Analyzer1`, it get last imported submission and compare it with new imported submission, if results with the same key have different value and last imported has value UNTESTED, analyzer save path to result and new value to object. Here is an example of that object in JSON format:

```
{
    "Categories": {
        "systemtap.apps": {
            ".\systemtap.apps\mysql.exp": {
                "mysql sdt app": "PASS"
            }
        }
    }
}
```

4.4 Vizualization

As was mentioned in chapter 2 ResultCloud use AngularJS for vizualization data. AngularJS is JavaScript MVC (Model-View-Controller) framework, every page has own controller, thus analyzer page must has it too. `AnalyzeController.js` is a file that contains controller for analysis result page. *analyze.html* is a page template. Some of the page, that contain several sort of data, builded 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 data for presenting.

Analyzer page would use only one component `analyzeOverview`.It wouldn't have any settings, and will support all plugins. Angular directory firstly get array of analyzing results for current submission, than for each analyzer find own Angular directory, which

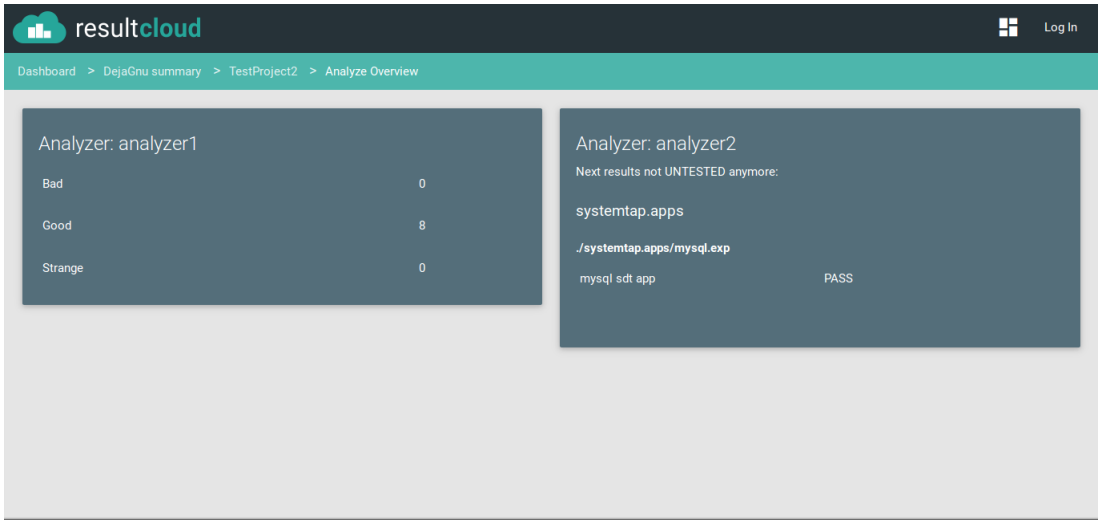


Figure 4.2: Implemented template

put analyzer data to template and present it. There some interesting part of code, how 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);
    }
}
```

Image4.2 present 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* puts vizualization data together into associative array the key analyzer ID and value analyze results, and return it.

4.5 Evaluation

During making bachelor's work was implemented analysis mechanism: `AnalyzerController` and two analyzers. All stuffs that was described in this chapter was implemented and functioning. Also mechanism are extendable, thus there is no problem to add new analyzers or new methods to it. Analysis results can be displayed in any format (list, diagram and etc.).

Chapter 5

Notification design

Notification mechanism must be flexible, and easy to extend. This chapter contains proposals for implementing notification mechanism. Here will consider mechanism's architectures, how implement notifiers better and notifications settings.

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 opposite to analyzers there are several types of notifiers. First type is public notifiers, it means that notifications would be sended into some shared or public resources, like *Twitter* for example. Private is means that it notifies each user separetly. According to this private notifiers must has settings, where user can set if he want to get notifications or not, and other options.

5.2 Notification Controller

Notification controller would has method for easy controlling notifications, which get all demanded fields, as *title*, *body*, *bodyShort* (for resources that accept only short messages), *adreses* (list of all adreses that must recieve notification) and according to them send notifications. Also `NotificationController` must provides methods for getting private and public `Notifiers`. When it included, it scans space for available `Notifiers`.

5.3 Notification settings

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

ResultCloud provides good tools for working with settings, like entities `TemplateSettings` and `TemplateSettingsItem`. `TemplateSettings` is for saving setting's template information into DB, such as 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 notifier it is useless. `Notifier` provides notification by itself, each notifier has own notification method, for example, by email, or Facebook.

5.4.1 Notifiers architecture

Every `Notifier` must have unique identifier(ID) for correct processing, by that ID `NotificationController` would identify notifiers, and it would have reference to notifier by this ID in settings template. `Notifier` must have one method for notification, and one method with settings. Each notifier has the same settings, thus that method can be picked out into some basic class, which would be inherited by notifiers classes. `Notifier` must have attribute that identifies it is like private or public notifier.

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

5.4.2 Notifier types

Within the confines of this bachelour's work, must be implemented these types of notifiers:

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

Chapter 6

Implementation of notification mechanism

This chapter contains 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 is not a plugin or any other ResultCloud kernel part, thus it also be in *extentions* folder, in own *notification* folder. `NotificationController` is also located in root directory. All notifiers are located in *notifiers* folder.

Notification starts only if analyzer return interesting result.

6.2 Settings

Basically all notifers have same setting, this settings would enable or disable notifier's notifications. But as mentioned in previous chapter, only private notifiers can use settings. Notification mechanism does not work with settings, because list of addresses and notifiers must be assamble by those who sending, notification mechanism only takes 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, to understand how set up settings better:

```
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

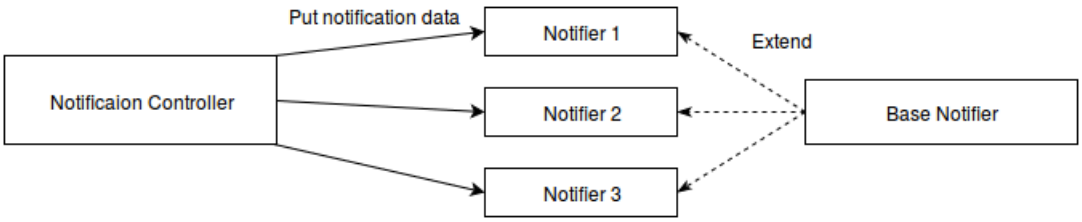


Figure 6.1: Implementation of notification mechanism

6.3.1 NotificationController

Like in case with `AnalyzerController`, `NotificationController` is static too, all methods are static, because there is no reason to create more than one class instance in application. Notification mechanism scheme6.1 showing how `NotificationController` connecting with other mechanism elements.

When `NotificationController` is included, it starts method *preLoad*, this method scans *notifiers* folder, include and assamble array with all available notifiers. There is exist a method for notifing *notify(title, body, bodyShort, to)*, where `title` - is title of notification message, `body` - the longest body text, `bodyShort` - short body of the message but no longer than 140 letters, `to` - is an array of all adresses with the key of notifier ID. Method *notify* calls notifier's function *notify* only for notifiers which have their IDs in `to` parameter's key.

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

6.3.2 Notifier

Every notifier must inherit `BaseNotifier` class with default settings, and if it is need to 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 contains boolean value, if notifier is public it contain *true*, otherwise *false*. The most important method is *notify*, it has same parameters such as method *notify* in `NotificationController`, except last address parameter, notifier would not get associative array, but gets a simple array with addresses.

6.3.3 Notify1

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

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

6.3.4 Twitter

`Twitter` is a public notifier that sends new twittes with some interesting information to twitter account. Now it connected to my account cyberbond95 . If analyzer would have interesting results it sends 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 [3]. It has MIT license. Page of `TwitterAPIExchange` with examples https://github.com/J7mbo/twitter-api-php.

6.3.5 RSS

`RSS` is a public notifier that create or update *rss.xml* file in root folder. *rss.xml* presents one news for each project, if `RSS` get news for already existing in *rss.xml* project, it updates news, otherwise it just create new one. Current actual version of RSS is 2.0, according to specification [4] second version is simplier than first.

`RSS` works with RSS by using `SimpleXML` that is default in most PHP versions. Here RSS example:

```
<?xml version="1.0"?>
<rss version="2.0">
<channel>
<title>ResultCloud News</title>
<link>http://result-cloud.org/</link>
<description>ResultCloud analysing results</description>
<language>en-us</language>
<docs>http://result-cloud.org/rss.xml</docs>
<item>
<title>Project TestProject has new interesting submission</title>
<description>Submission with id 22 has interesting results according
to [analyzer1] analyzers, for more information go there
http://corly.local/#/project/1/analyze/22</description>
<link>http://corly.local/#/project/1/analyze/22</link></item>
</channel>
</rss>
```

6.4 Evaluation

Notification mechanism support adding new notifiers (if notifier was implemented like it was described in this chapter) and extending settings. `NotificationController` and three type of notifiers was implemented. One of notifiers is private email notifier. Other two notifiers are public.

During testing was found two problems: email sending on my computer get a lot of time, Twitter do not allow to send two twits with the same content. First problem is a local and related with software on my computer. Second problem is actual only when debuging, because in production content always will be different, in production current ID value for table never be discard, thus url in twit always be unique.

Chapter 7

Conclusion

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

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

Mechanism can be extending by adding new analyzers and notifiers. Also can be extending notification mechanism by adding new settings, for example: user can choose about which analyzers he wants get notification. Extend analyzer mechanism, add to each analyzer configuration file, and make it switchable like components in ResultCloud.

Bibliography

[1] Filip Matys. *Webový nástroj pro správu výsledků dlouhodobého testování*. Master's thesis, FIT VUT v Brně, 2014. [Online; cit. 16.5.2016].

[2] AngularJS team. *AngularJS Official Documentation*. [online], 2016. URL: https://docs.angularjs.org/guide/introduction.

[3] Twitter. *Twitter API documentation*. [online], 2016. URL: https://dev.twitter.com/overview/documentation.

[4] Dave Winer. *RSS 2.0 Specification*. [online], 2003. URL: http://cyber.law.harvard.edu/rss/rss.html.

Appendices