judgels Documentation

Release 0.8.0

ia-toki

Jun 19, 2017

Contents

1 Overview of Judgment Angels 3
2 Operator’s Guide 7
3 Administrator’s Guide 31
4 Introduction for Judgels developers 53
5 Overview 63
6 Operator’s Guide 65
7 Administrator’s Guide 67
8 Developer’s Guide 69

Welcome to Judgment Angels (**Judgels**) documentation! This documentation is divided into four sections:

CHAPTER 1

Overview of Judgment Angels

What's Judgment Angels?

Judgels (**Judgment Angels**) is a set of modular applications for educational programming purposes, such as:

- creating and storing programming problems,
- holding programming contests,
- holding learning courses for programming,
- discussing programming problems in forum,
- and many more.

Each user uses a single account to access all services.

This project was initiated by Ikatan Alumni Tim Olimpiade Komputer Indonesia (English: Indonesia Computing Olympiad Alumni Association).

Judgels applications

Judgels consists of several applications that work with each other. Each application has a codename after a Greek archangel name.

At the moment, there are eight applications in Judgels:

Jophiel (Single Sign-On) Authenticates and authorizes users in other Judgels applications. The motivation is that a user should have only a single account accross all Judgels applications. It uses OpenID Connect protocol.

Sealtiel (Message Gate) The bridge that handles asynchronous messages between Judgels applications. Currently it is only used for delivering grading requests and responses between Gabriel and other Judgels applications that need grading.

Sandalphon (Repository Gate) Stores programming resources, for example, problems and lessons. The resources can then be used by other Judgels applications, like Uriel and Jerahmeel.

Uriel (Competition Gate) Holds programming contests.

Jerahmeel (Training Gate) Holds programming training and problemsets archive.

Raguel (Forum) Place for discussions.

Gabriel (Grader) Grades programming submission requests from Sandalphon/Uriel/Jerahmeel in a sandbox, then sends back the results via Sealtiel.

Michael (Alchemy Gate) [EXPERIMENTAL] Monitors machines used for running other applications.

The applications are designed to be modular. For example, multiple instances of Uriel can share the same Sandalphon and Jophiel instance. They are also designed to be distributed: the required application instances do not have to be installed in one single machine. We can install one application in one machine and some others in other machines.

Judgels applications are still being heavily developed and do not have any stable releases yet. Anyone can try at their own risks.

Repository

Judgels is open source. The whole source code is hosted in GitHub: https://github.com/judgels.

Credits

This project is currently being developed by the following people, each in alphabetical order.

Maintainers

- Ashar Fuadi (@fushar)
- Jordan Fernando (@dragoon20)
- Petra Novandi Barus (project manager) (@petrabarus)

Contributors

- Bagus Seto Wiguno (@bswig)
- Deny Prasetyo (@jasoet)
- Ivan Hendrajaya (@ivanhendrajaya)
- Inggriani Liem

- Adi Mulyanto
- Arief Widhiyasa
- Derianto Kusuma
- Brian Marshal
- William Gozali

Public contributions are welcome. Please consult the Developer's Guide on how Judgels works in details.

License

Judgels is licensed using GNU GPL version 2.

CHAPTER 2

Operator's Guide

This guide is intended for people who want to operate and manage existing Judgels applications.

Table of Contents

Single Sign-On

TBA.

Repository Gate

Repository Gate is a place for storing resources used by other Judgels applications. There are two kinds of resources: **problems** and **lessons**. A problem is an HTML document that shows question to be solved by *submitting* a solution to it. A lesson is an HTML document that shows learning materials only.

A typical workflow for you as a Repository Gate's operator is as follows:

- You create new problems/lessons. For problems, you will also have to specify some configurations. For example, for programming problems, you have to configure test cases.
- You (or another operator) **share** them to Repository Gate's **clients**, such as Competition Gate or Training Gate. The administrator should have set up the clients.
- You (or another operator) add the shared problems/lessons in a contest (in Competition Gate) or course/session (in Training Gate).

You will learn more about lessons and problems, common features in problems and lessons, and how to share resources to the clients in the next sections.

Table of Contents

Managing problems

In this section, you will learn how to write a problem and configure it.

Problem types

There are two types of problems that are supported in Repository Gate: **programming** problems and **bundle** problems.

A programming problem is a typical competitive programming problem which has a set of test cases used for grading contestant's solution.

A bundle problem is a non-programming problem each containing several items. Currently, an item can only be a simple statement or a multiple-choice question.

Problems components

When creating a new problem, you have to specify these components.

Type The problem type.

Slug A "code" or "identifier" for the problem. The format is dash-separated words each consisting of lowercase letters, numbers, or both. For example: **a-plus-b**, **tree-reconstruction**, **2sat-construction**, etc.

Additional Note Any additional note. For example, problem tags, contests that have used the problems, etc.

Initial Language The initial language of the problem statement. Later, you can add other languages.

Next, please see *Common resources features* for documentation of common features in resources.

In the next sections, you will learn about specific configurations for different types of problems.

Managing programming problems

A programming problem here is a typical **blackbox** competitive programming problem. Blackbox means that contestant submissions are graded based on the outputs on the provided **test cases**. The submission source code is not checked.

Currently, the allowed programming languages are Pascal, C, C++, and Python 3 only.

The specific configuration for programming problems can be found in **Grading** tab.

Test data concepts

Test cases

A test case is the smallest unit of grading data. It consists of several files. Test cases that appear in the problem statement are called sample test cases.

When the contestant program is evaluated against a test case, the **verdict** can be one of the following, from the lowest **priority** to the highest **priority**:

- **AC** (Accepted): the contestant program is considered correct.
- **OK X** (OK): the contestant program is partially correct, with X points.
- **WA** (Wrong Answer): the contestant program is considered incorrect.
- **RTE** (Runtime Error): the contestant program crashed; may be because it exceeded the memory limit.
- **TLE** (Time Limit Exceeded): the contestant program did not stop within the time limit.
- **SKP** (Skipped): the contestant program was not run at all. This will be explained in problems with subtasks.

Time limit

The maximum time (not wall time) a contestant program can run on a test case.

Memory limit

The maximum memory a contestant program can consume. Stack size is only limited by this memory limit.

Source code limit

Currently it is hardcoded to **300 KB**.

Programming problem types

Blackbox problems are classified based on two aspects: **evaluation methods** and **scoring methods**. Based on evaluation methods, there are four types: **batch**, **interactive**, **output-only**, and **functional**. Based on scoring methods, there are two types: **with subtasks** and **without subtasks**.

Batch problems

This is the standard and most common programming problem type. For each test case, the contestant program reads the input file and produced an output file. (In the program perspective, it reads from standard input and writes to standard output.) By default, the test case verdict is AC if the produced output file match exactly with the inteded output file, or WA otherwise.

It is also possible that in some problems, multiple answers are considered correct, or even partially correct. In order to support that, it is possible to write a custom **scorer** that decided the test case verdict. See the **Helper files** section for more details.

Interactive problems

For each test case, the contestant program interacts with the so-called **communicator** program. There is only test case input file. The communicator program also decides the test case verdict, based on the interaction result. See the **Helper files** section for more details on writing communicator program.

Output-only problems

For this problem, contestants do not submit their source code. Instead, they are given the input files (X.in), and they submit a zipped file containing the output files (X.out). Each of the submitted output file will be then checked against the judge's output file using simple diff or custom scorer, similar to what it is done in batch problems.

Functional problems

This type of problem is explicitly created to support IOI problems since 2010. Here, contestants may submit more than one source files. In each source file, instead of reading the input froms stdin, they have to implement a function, and the input will be given as function arguments. At the moment, only C++11 language is supported.

A functional problem consists of one or more submission "slots", which we will call "keys". For each key **X**, contestants must write a source file **X.cpp**, which should implement a function defined in **X.h**. For example, in IOI 2010 Cluedo, we have one key: **cluedo**, whereas for IOI 2010 Saveit, we have two keys: **encoder** and **decoder**. There are also two special files: **grader.cpp** and **grader.h**. These files are used both for grading and for contestants' testing.

Usually, here is how to compile contestants' source files for testing:

```
g++ -o grader grader.cpp <space-separated list of source files .cpp>
```

For example,

```
g++ -o grader grader.cpp encoder.cpp decoder.cpp
```

How to set up

- Modify **grader.cpp** so that it outputs test case verdict in the format as explained in **Helper files**.
- Upload the following files as helper files:
  - grader.cpp
  - grader.h
  - All X.h, where X is a key (for example, {cluedo.h}, {encoder.h, decoder.h}
- In the **Keys** field in Grading Config configuration, fill in comma-separated keys. For example: **cluedo** or **encoder,decoder**.

Problems without subtasks

The score of a submission to this problem is simply the sum of test case scores.

The score of a test case is defined based on the test case verdict:

- If the verdict is **AC**, then the test case score is **100.0 / (number of test cases)**.
- If the verdict is **OK X**, then the test case score is **X**.
- If the verdict is anything else, then the test case score is **0**.

The verdict of a submission is the verdict with the highest priority among the test case verdicts.

Problems with subtasks

This type of problem introduces new concepts: **subtasks** and **test groups**.

Subtasks A subtask is a set of constraints. A problem can have multiple subtasks (multiple set of constraints), to give nicer score distribution. For example, a problem can have 3 subtasks with these sets of constraints:

1. N = 1, 1 <= K <= 100
2. 1 <= N <= 100, K = 1
3. 1 <= N <= 100, 1 <= K <= 100

A test case should be assigned to a subtask if the test case satisfy all constraints in the subtask.

Test groups A test group is a set of test cases that are assigned to the same set of subtasks. Test groups are introduced to simplify the organization of assignment of test cases and subtasks. For example, for the above problem, we can create 4 test groups as follows:

1. Consists of only one test case N = K = 1. Assign it to subtasks {1, 2, 3}.
2. Test cases that satisfy N = 1; 2 <= K <= 100. Assign them to subtasks {1, 3}.
3. Test cases that satisfy 2 <= N <= 100; K = 1. Assign them to subtasks {2, 3}.
4. Test cases that satisfy 2 <= N, K <= 100. Assign them to subtasks {3}.

The score of a submission is the sum of the score of each subtask.

The score of a subtask is:

- the points assigned to the subtask, if each test case assigned to the subtask has **AC** verdict, or
- **0**, if at least one test case assigned to the subtask does not have **AC** verdict.

The verdict of a subtask is the verdict with the highest priority among the test case verdicts.

The verdict of a submission is the verdict with the highest priority among the subtask verdicts.

If a test case will not affect the verdicts of all subtasks assigned to it anymore, the test case will be skipped (has Skipped verdict). For example, if a test case is assigned to Subtask 1, but there has been a test case in Subtask 1 that is not Accepted, then that test case will be skipped.

Grading engines

Based on the classifications above, there are 4 types of programming problems supported in Sandalphon. The type is referred to as **grading engine**.

- Batch
- Batch with subtasks
- Interactive
- Interactive with subtasks

Helper files

These files should be uploaded to the Helpers section in grading configuration. You must upload the **source code**, not the executable program. The helper files mostly decide test case verdicts.

The test case verdict takes one of the following format:

- Accepted

  ```
  AC
  <info>
  ```

- OK

  ```
  OK
  X <info>
  ```

  where **X** is the score. Can be a floating-point value.

- Wrong Answer

  ```
  WA
  <info>
  ```

In all cases, **<info>** is an additional info which will be given to the contestants in the submission result details. For example, in a binary search interactive problem, the additional info may be the number of guesses the contestant program gave. If you don't want to give additional info, just omit it. In AC and WA verdicts, just omit the second line altogether.

Scorer

A scorer is a C++ program which decides the verdict of a test case in batch problems.

The scorer will receive the following arguments:

- argv[1]: test case input filename
- argv[2]: test case output filename
- argv[3]: contestant's produced output filename

The scorer must print the test case verdict to the **standard output (stdout)**.

Here is an example scorer program which gives AC if the contestant's output differs not more than 1e-9 with the official output.

```cpp
#include <fstream>
#include <iostream>
#include <algorithm>
using namespace std;

int wa() {
    cout << "WA" << endl;
    return 0;
}

int ac() {
    cout << "AC" << endl;
    return 0;
}

int main(int argc, char* argv[]) {
    ifstream tc_in(argv[1]);
    ifstream tc_out(argv[2]);
    ifstream con_out(argv[3]);

    double tc_ans;
    tc_out >> tc_ans;

    double con_ans;
    if (!(con_out >> con_ans)) {
        return wa();
    }

    if (abs(tc_ans - con_ans) < 1e-9) {
        return ac();
    } else {
        return wa();
    }
}
```

Communicator

A communicator is a C++ program which interacts with the contestant program in interactive problems, and then decides the verdict of a test case.

The communicator will receive the following argument:

- argv[1]: test case input filename

During the interaction, the communicator can read the contestant program's output from the **standard input (stdin)**, and can give input to the contestant program by writing to the **standard output (stdout)**. Make sure the communicator flushes after every time it writes output.

Ultimately, the communicator must print the test case verdict to the **standard error (stderr)**. Note that (currently) the interaction is not guaranteed to stop after the verdict has been output, the interaction may exceed the time limit if neither it or contestant program stops.

Here is an example communicator program in a typical binary search problem. In this example, the organizer wants that the number of guesses be output in an AC verdict.

```
#include <fstream>
#include <iostream>
using namespace std;

int wa() {
    cerr << "WA" << endl;
    return 0;
}

int ac(int count) {
    cerr << "AC" << endl;
    cerr << "guesses = " << count << endl;
    return 0;
}

int main(int argc, char* argv[]) {
    ifstream tc_in(argv[1]);

    int N;
    tc_in >> N;

    cout << N << endl;

    int guesses_count = 0;
    while (true) {
        int guess;

        cin >> guess;
        guesses_count++;

        if (guesses_count > 10) {
            return wa();
        } else if (guess < N) {
            cout << "TOO_SMALL" << endl;
        } else if (guess > N) {
            cout << "TOO_LARGE" << endl;
        } else {
            return ac(guesses_count);
        }
    }
}
```

Language restriction

You can limit which programming languages are allowed for a submission to a problem, in the **Language Restriction** subtab.

Managing bundle problems

A bundle problem is a non-programming problem. Here is a sample bundle problem:

BEGIN SAMPLE BUNDLE PROBLEM

[**A**] Municipal Olympiad in Informatics

[**B**] Please answer all questions correctly!

[**C**]

This is a description for numbers 1 through 2.

We define f(N) = N * (N-1) * (N-2) * ... * 1.

[**D**] What is the value of f(5)?

1. 5
2. 20
3. 120

[**E**] What is the value of f(1)?

1. 1
2. 0

3. 2

END SAMPLE BUNDLE PROBLEM

Bundle problem components

As you can see, a bundle problem consists of these components:

Title The problem title. In the above sample problem, the problem title is "Municipal Olympiad in Informatics" (A).

Text The problem text, usually serves as the instruction for doing the questions. In the above sample problem, the problem text is "Please answer all questions correctly!".

Items Next, a bundle problem consists of one or more items. When adding an item to a bundle problem, you have to specify its **item slug (meta)**. This slug serves the same way as a problem slug, but for items.

The above sample problems has three items: (C), (D), (E). Their slugs could be "factorial-desc", "factorial-1", and "factorial-2".

Bundle item types

An item can be of one of the following types.

Item type: statement

This item is just an HTML text that must stand on its own because it is required by one or more questions. Typically, this contains a background story or some definitions that are referred to by multiple questions below it. It is a passive item; you cannot "answer" this item. The above sample problem has a statement: (C).

Item type: multiple-choice question

As the name says, this is a multiple-choice questions. A multiple-choice question has the following components:

Statement The question statement.

Score Points added for answering this question correctly. If the question is not answered, the score will be 0.

Penalty Points deducted for answering this question incorrectly.

Choices The answer choices. Each choice has **alias** (for example, "a", "b", etc.) and **content** (the answer itself). You can specify one or more choices as correct answer(s).

The above sample problem has two multiple-choice questions: (D) and (E).

Lessons

A lesson is a passive HTML document that shows learning materials.

You can write the learning materials directly using HTML, or embed external PDFs.

Lesson components

When creating a new lesson, you have to specify these components.

Slug A "code" or "identifier" for the lesson. The format is dash-separated words each consisting of lowercase letters. For example: **dynamic-programming**, **advanced-greedy**, etc.

Additional Note Any additional note. For example, lesson tags, trainings that have used the lessons, etc.

Initial Language The initial language of the lesson statement. Later, you can add other languages.

Next, please see *Common resources features* for documentation of common features in resources.

Common resources features

In this section, you will learn about common features that are present in both lessons and problems.

Managing statements

Writing title and text

Don't forget to specify resource title, since you only specified the slug when you created the resource.

For writing the resource text, you are provided with a WYSIWYG editor.

Inserting images

You can insert images to the statement. Upload the image as media file in the **Media** subtab, and then insert it to the statement by adding an image with this URL: **render/<imageFilename>**.

Adding other languages

You can have multiple versions of the statements, based on language. You can add new languages in the **Languages** subtab. **Default language** is the default language shown to the user.

You can also disable a language. When viewing the statement, users can switch language to their preferences

Managing partners

You can share the privilege to modify the resource you created to other users (called **partners**). Add the username of the user you want to share, in the **Partners** subtab. You can specify the permissions you want to grant.

Managing versions

Every changes to a resource will be tracked in a version control. Every time you consider that you have finished making a change, you have to **commit** your change so that the changes take effect. When you commit, one of the following events will happen:

- The commit succeeded. Your changes will be recorded.
- New commits have been made since you started editing. You will be prompted to update your local working copy. Click **Update Working Copy**. One of the following events will happen:
  - Update working copy succeeded. You can continue committing.
  - Update working copy failed. Your changes conflicts with the previous commits. Currently, there is no solution. You have to remember/save your changes somewhere and then click **Discard Local Changes** to discard your changes and load the latest version.

Note that you can also update working copy or discard local changes any time during your modification.

Sharing resources to clients

If a Repository Gate's client wants to use a resource, perform these steps.

1. Enter the corresponding resource (problem/lesson).
2. Click the **Clients** tab.
3. Choose the client, and then click **Create New**.
4. View the client. You will see the resource JID and resource secret, for that particular client. Using that pair of JID and secret, you can add the resource to contests/sessions/etc.

Competition Gate

Competition Gate is a place for holding contests. As of now, only programming contests are supported. Specifically, only ICPC-style and IOI-style programming contests are supported. Contests with bundle problems are not (yet) supported.

First, let's learn about operator roles in Competition Gate. There are three roles: **admins**, **managers**, and **supervisors**. Admin role is global throughout Competition Gate, while manager and supervisor roles are specific to a certain contest.

Admin

- Can create new contests.
- Can assign managers to a contest.
- Can remove managers from a contest.
- Can do all managers' and supervisors' tasks.

Manager

- Can edit contest's configurations.
- Can assign supervisors to/from a contest.
- Can remove problems from a contests.
- Can remove contestants from a contest.
- Can do all supervisors's tasks.

Supervisor Can supervise a contest based on the given permissions. For example: posting announcements, answering clarifications, adding problems, adding contestants, adding files, etc. It cannot perform destructive tasks such as removing problems and contestants.

For simplicity, by "contest manager" we mean contest manager or admin, and by "contest supervisor" we mean contest supervisor, contest manager, or admin.

You will learn how to do contest-related tasks in the next sections.

Table of Contents

Managing contests

A contest can **only** be created by an operator with **admin** role. After the contest has been created, the admin can then assign one or more **managers** to it. The managers or the admin themselves will configure the contests.

Contest components

When you create a new contest, you will be presented with the following fields.

Name The contest name.

Description The contest description that will be shown to the public. Can contain promotion text so that users register to this contest, etc.

Style Currently two styles are supported: ICPC-style and IOI-style contests. Further details will be given in the next sections.

Contest managements aspects

Contest managements refer to configuring contest before the contest starts and supervising contest during the contest.

Contest managements are divided into several aspects. Each aspect can consist of configuration units called **contest modules**. Each module can be enabled or disabled (almost) independently. Some contest modules also have module configurations that can be further specified when the modules are enabled.

Here is an overview of contest management aspects:

Managing contest style How to configure settings related to the contest style, such as ICPC time penalty etc, allowed programming languages, etc.

Managing contest scope How to control who can access the contest, can users register to the contest, etc.

Managing contest times How to specify the duration of the contest, can the contest be started virtually, etc.

Managing contest people How to add people with different roles to the contest: managers, supervisors, and contestants.

Managing contest problems How to add problems from Repository Gate to the contest.

Managing contest teams How to make the contest team-based.

Managing contest triggers How to control who can start the contest.

Managing contest passwords How to set passwords in the contest.

Managing contest announcements How to use announcements in the contest.

Managing contest clarifications How to enable and use clarifications in the contest.

Managing contest submissions How to view and regrade submissions in the contest.

Managing contest scoreboards How to enable scoreboards in the contest, set scoreboard freeze time, etc.

Managing contest files How to upload public files in a contest.

Locking contests How to lock a contest after it ends.

Finally, the full references of all contest modules are available in *Contest module references*.

Managing contest style

A contest style will determine the allowed problem types and how the scoreboard is computed. Currently, two contest styles are supported: ICPC-style and IOI-style contests.

For both styles, you can specify programming languages allowed in the contest. For each problem in the contest, the final allowed languages are the **intersection** of the set of allowed languages of the problem (specified in Repository Gate) and the set of allowed languages in the contest.

Similar to Repository, only Pascal, C, C++, and Python 3 languages are currently supported.

ICPC-style

Contests using usual ACM-ICPC rules. A submission is considered to have "accepted" verdict if its score is at least 100. This "normalization" is useful if you want to use "IOI-style" problems (that have subtasks) in an ICPC-style contest.

In this style, you can specify how much time penalty will be incorrect for each non-accepted submissions in a problem that is finally accepted.

IOI-style

Contests using IOI rules. Ranking is based on total scores only. No other configurations are present for this style.

Managing contest scope

This refers to how a user can join a contest (i.e., become a contestant).

Public

In this scope, everyone can join the contest. To enable public scope, simply disable **Limited** module.

This scope is further divided into two types.

Public with registration

A user need to register before they can join the contest.

To enable registration, enable **Registration** module. In the module configuration, you can specify registration start and end times, and the maximum number of allowed registrants. By default, a registered user can automatically join the contest when the contest starts, unless you enable the **Manual Approval** option in the module configuration.

If you want to gather users' organizations or institutions in the contest, simply enable **Organization** as well. Then, when a user registers, they must provide their organization.

Public without registration

A user need not register before they can join the contest. Typically, this scope is used for really public contests; for example, an introductory contest containing sample introductory problems.

Simply disable the **Registration** module.

Private

In this scope, supervisors have to manually add contestants to the contest before they can join the contest. The contest won't be shown in the contests list for guests or non-contestants.

To enable private scope, disable **Registration** module and enable **Limited** module.

Managing contest times

These are things that can be configured related to contest times.

Eternal

Contests that are available forever. Disable **Duration** module.

Fixed-time

Contests that have specific begin and end times. Enable **Duration** module. You can then specify begin time. For convenience, you don't specify the end time, but the contest duration. This way, when the contest needs to be delayed, you only have to adjust the begin time.

Virtual

Contests that can be started by each contestant at their preferred time. Enable **Virtual** module. You can then specify virtual duration time. A contestant can start the contest anytime between the contest begin and end time. Then, the contest will run for that contestant for the specified virtual duration time. The contest will finish for that contestant when the contest ends or the contestant runs out of virtual duration time, whichever happens first.

Virtual contests are useful for conducting an online contest which have specified duration, but does not have to start at the same time for every contestant.

Paused

If **Paused** module is enabled, then the contestants cannot "do" the contest, i.e. submittion solutions or creating clarifications. This is useful when there is something wrong with the grader and you want to fix it, therefore you must stop any submissions.

The contest will end at the scheduled time – no additional time is added if you pause contests.

Managing contest people

As described *here*, there are three roles for operator: **admins**, **managers**, and **supervisors**.

Managing managers

Admins can assign or remove managers in the **Managers** tab in the contest.

Managing supervisors

Managers can assign or remove supervisors in the **Supervisors** tab in the contest. A set of permissions can be given for each supervisor:

Supervising announcements Posting and disabling announcements.

Supervising problems Adding and disabling problems.

Supervising submissions Viewing all submissions and regrading submissions.

Supervising clarifications Viewing all clarifictions and answering clarifications.

Supervising contestants Adding and disabling contestants.

Supervising teams Forming teams from contestants.

Supervising scoreboards Viewing official scoreboard.

Supervising files Uploading files.

Managing contestants

Supervisors can add and disable contestants in the **Contestants** tab in the contest.

Only managers can remove contestants.

Methods for adding people

People with any role can be added using two interfaces: manually adding persons one-by-one, and adding multiple persons at once using a text file.

If you want to use the latter method, you can upload a plain text file containing newline-separated usernames of the people you want to add.

Managing contest problems

Problems can be added to the contest in the **Problems** tab. Only managers can remove problems.

When adding a problem, you will be presented with the following fields.

Alias Alias for the problem in the contest. For example: **A**, **B**, **C**, etc.

Problem JID, Problem Secret The credentials you obtained from Repository Gate after you share the problem to this Competition Gate. See *Sharing resources to clients*.

Submissions Limit Maximum number of submissions allowed. Set to 0 if no limit.

Status Can be one of the following:

- **Open**: problem is visible and used in the contest.
- **Closed**: problem is visible and used in the contest, but submissions are disabled.
- **Unused**: problem is not visible and not used in the contest.

Managing contest teams

A team-based contest can be conducted in Competition Gate. To enable team-based contest, enable **Teams** module. Teams can be assigned in the **Teams** tab.

A team consists of

- Team name
- Team logo (optional)
- One or more coaches
- One or more members as the contestants

A team-based is more or less similar to usual a contest. The only major difference is that coaches:

- Can see the scoreboard entries for their contestants.
- Can translate clarifications from their contestants.
- In virtual contests, coaches can start the contest for their contestants.

It is possible that a contestant does not get assigned to a team. A contestant can be assigned to at most one team. A user, however, can be a coach of multiple teams.

Managing contest triggers

For a contest that is both virtual and team-based, you can specify that only coaches can start the contest for their own contestants. An example that needs this feature is Asia-Pacific Informatics Olympiad (APIO) contests.

Simply enable **External Trigger** module, then select **Coach** in the configuration.

The coaches can then start the contest via **Teams** tab. They can click the start button to start the contest. Once the contest has started, it can't be undone.

Managing contest passwords

A contest can be given a password that the contestants must type before they can enter the contest. This is useful for onsite contests, so that contestants cannot cheat by sharing their Single Sign-On passwords. Enable the **Password** module to activate this feature.

You can generate the passwords in the **Password** tab. The password for each contestant will be different from each other. You can re-generate all passwords or individual passwords.

Managing contest announcements

You can post announcements in the contest. Contestants can see the posted announcements. Contestants will be notified when new announcements are posted. The number of unread announcements will be shown next to the **Announcement** tab.

Managing contest clarifications

To enable clarifications in the contest, enable the **Clarifications** module.

If you want that clarifications can only be made for a specified duration after the contest begins, enable the **Clarifications Time Limit** module. You can then specify the duration since the contest begins.

Clarifications behave differently for contestants, supervisors, and coaches.

For contestants Contestants can make clarifications during the contest or during the clarification time window. They can specify the problem they want to clarify, or just general clarification.

For coaches In a team-based contest, coaches can translate the clarifications made by their contestants.

For supervisors Supervisors can answer contestants' clarifications.

Managing contest submissions

Supervisors can view all submissions. Supervisors can also regrade individual submissions or all submissions.

Managing contest scoreboards

To enable scoreboard in the contest, enable the **Scoreboard** module. In the module configuration, you can set that the scoreboard will be **incognito**, i.e. contestants can only see their own scores in the scoreboard. Supervisors can always see full scoreboard.

If you want that the scoreboard can be frozen, enable the **Freezable Scoreboard** module. Frozen scoreboard can only be used in contests that are non-virtual and have fixed times. In the module configuration, you can set the freeze time that is specified as duration before the contest ends. After that, you can unfreeze the scoreboard from the module configuration.

Managing contest files

You can upload files to the contest and have the links available in announcements. Enable the **Files** module. Then, upload the file in the **Files** tab. You can then insert the link in announcements as a link with the url **download/<filename>**.

Locking contests

After a contest ends, admins can "lock" the contest, by clicking the **lock** button beside the contest name. This will prevent any future modifications to the contest. Basically, if a contest is locked, it is considered to be "archived".

Admins can also unlocked locked contests.

Contest module references

Here is a complete reference of all available contest modules.

Limited

Prevents the contest from being shown and being registered by public users. Contestants must be expicitly added by supervisors.

Conflicting modules

- Registration

Registration

Allows public users to register to the contest.

Configurations

Registration start time The earliest time users can register.

Registration duration How long users can register since the start time.

Requires manual approval? Whether supervisors must manually approves the registrants.

Max registrants Maximum number of registrants. Set to 0 if unlimited.

Conflicting modules

- Limited

Organization

When enabled, users must enter their organization/institution when registering for the contest.

Required modules

- Registration

Duration

Sets a fixed time of contest time.

Configurations

Contest begin time When the contest begins.

Contest duration Duration of the contest.

Virtual

Allows users to start the contest at their own preferences of starting time.

Configurations

Virtual contest duration Maximum time a user can do the contest.

Conflicting modules

- Freezable Scoreboard

Paused

Pauses the contest. Users cannot submit or make clarifications. No additional time is given.

Supervisors

Allows supervisors to be added in the contest.

Teams

Make the contest team-based.

External Trigger

Make a virtual, team-based contest to be able to be started by the external person.

Configurations

Contest will be started by Who can trigger the contest. Currently can only be set to coach.

Required modules

- Virtual
- Teams

Password

Set passwords that must be typed by the contestants before they can enter the contest.

Clarifications

Enable clarifications in the contest.

Clarifications Time Limit

Sets how long contestants can make clarifications.

Configurations

Clarification duration How long contestants can make clarifications, since the beginning of the contest.

Required modules

- Clarification
- Duration

Scoreboard

Enable scoreboard in the contest.

Configurations

Incognito scoreboard? Whether a contestant can only see their own scores in the scoreboard.

Freezable Scoreboard

Allow the scoreboard to be frozen.

Configurations

Scoreboard freeze time When the scoreboard will freeze, specified as duration before the contest ends.

Scoreboard has been unfrozen? If this is checked, scoreboard will be unfrozen.

Required modules

- Duration
- Scoreboard

Conflicting modules

- Virtual

Files

Allow public files to be uploaded to the contest.

Training Gate

TBA.

Alchemy Gate

TBA.

Forum

TBA.

CHAPTER 3

Administrator's Guide

This guide is intended for people who want to set up working Judgels application instances.

Table of Contents

Setup

A Judgels application is installed by cloning and building the source code from GitHub. A Judgels application may depend on other Judgels applications to work. Judgels is designed to be distributed, so:

- More than one applications can be installed on one machine.
- Two applications can work together regardless of whether or not they are on the same machine.

Requirements

Theoretically, all Judgels applications except Gabriel can run on any operating system. However, we have not tested thoroughly on Windows. Therefore, we strongly recommend that you use UNIX-based operating systems (Linux or OS X). Also, throughout this documentation, it will be assumed that you are using Linux or OS X.

The following programs must be installed on any machine that hosts any Judgels application:

- Oracle Java 8
- Git
- Python 3

Specific requirements for each Judgels application will be mentioned on the respective application page.

Installing main repository

Judgels consists of many repositories. Create a directory that will store Judgels repositories. We will denote it as Judgels home. There must be only one Judgels home even though more than one applications will be installed on the machine.

First, clone the main Judgels repository. This repository will act as a gate to the other Judgels repositories.

```
cd <your-Judgels-home>
git clone https://github.com/judgels/judgels
```

Then, we will install Judgels terminal script that will enable many convenience commands in the terminal. Open your **~/.bashrc**, and add the following lines.

```
export JUDGELS_HOME=<your-Judgels-home>
alias judgels="python3 $JUDGELS_HOME/judgels/scripts/terminal.py"
```

Restart your terminal to activate the script.

More information about this command-line tool can be found here: *Command-line tool*.

Installing Activator

All Judgels applications use Typesafe Activator as the wrapper for SBT. Activator is needed for compiling, running, starting, and distributing the applications.

First, download Activator:

```
cd $JUDGELS_HOME
judgels/scripts/download-activator.sh
```

Then, add this line to your **.bashrc**:

```
export PATH=$PATH:~/activator
```

Try running Activator:

```
activator
```

If it ran, then Activator has been installed successfully.

Installing database

All Judgels Play applications requires a connection a working MySQL database server. Install a MySQL server. Then, create a database named **judgels_<app>** (note the underscore, not dash) for each Judgels Play application <app> you want to install. For example: **judgels_jophiel**.

Installing application

If you want to install a Judgels application <app>, run

```
judgels pull <app>
```

(For example: `judgels pull jophiel`.)

This will clone the repositories of the application and all its dependencies. Then, you should follow the instruction on the respective application page. But, first modify the configuration files as explained in the next section. Some configuration keys are common to all application, so the instruction is provided on this page.

Modifying configuration files

All Judgels applications require editing configuration files. This section will explain the configuration keys that are common to all applications, except Gabriel. If you are installing Gabriel, skip this section. Specific configuration keys are explained in the respective application section.

First, copy the default configuration files. Run these commands in the application repository.

```
cp conf/application_default.conf conf/application.conf
cp conf/db_default.conf conf/db.conf
cp conf/akka_default.conf conf/akka.conf
```

Here are the configuration keys you need to modify. Note that if a key is not listed here or on the specific application page, then you don't need to modify its value.

Application configuration

The configuration file to modify is **conf/application.conf**.

general.title The displayed title/name of application. For example: "Public Competition Gate".

general.copyright The displayed copyright/institution name that hosts the application. For example: "XXX University".

general.canonicalUrl The same value as **<app>.baseUrl**.

play.crypto.secret Play framework's secret key for cryptographics functions. The default value must be changed for security. See https://www.playframework.com/documentation/2.4.x/ApplicationSecret for more details.

play.http.session.secure Set to true if you use HTTPS.

<app>.baseUrl The base URL address of the application. Do not include trailing slash. For example: "http://localhost:9001". ("http://localhost:9001/" is wrong.)

<app>.baseDataDir The absolute path of a local directory that hosts this application's data files. For example: "/home/user/judgels/data/jophiel".

seo.{metaKeywords, metaDescription} SEO meta keywords and description.

google.analytics.* See optional features section below.

google.serviceAccount.* See optional features section below.

redis.* See optional features section below.

Database configuration

The configuration file to modify is **conf/db.conf**.

url Fill it with database URL. If you install MySQL in localhost, the value should be "jdbc:mysql://localhost/judgels_<app>".

username Database's username.

password Database's password.

Akka configuration

Akka is used for concurrency management. It is safe to use the default configuration without modification.

Setting up optional features

Google Analytics reporting

This corresponds to **google.analytics.*** keys in **application.conf**.

Set **use** to true to enable Google Analytics reporting.

If used, set **id** to the Google Analytics ID. You should have different views for each Judgels Play application. You will have a unique view ID in Google Analytics. Set **viewId** to that ID.

Google Service Account

This corresponds to **google.serviceAccount.*** keys in **application.conf**.

Set **use** to true to enable pulling Google Analytics reporting.

If used, go to Google Developers Console and register a credentials. Set the appropriate values then. Currently it is only used for showing current active users.

Note: It is an **EXPERIMENTAL** feature as there is limit on the number of requests, so it is usually turned off now.

Redis

You can use in-memory cache Redis for performance.

To use Redis:

- Set up a working Redis installation.
- Modify **redis.*** keys in **application.conf** accordingly.
- In **application.conf**, add two new modules:

```
enabled += "com.typesafe.play.redis.RedisModule"
enabled += "org.iatoki.judgels.play.jedis.JedisModule"
```

Note: It is an **EXPERIMENTAL** feature. The only things that are cached are queries to IDs.

Running Judgels Play applications

After the installation and configuration, we can run Judgels play applications in two modes.

Development mode

Run the `judgels run <app>` command. This mode is intended for development environment. Classes will be automatically recompiled if there are changes in the corresponding source files, without having to restart the application.

Production mode

In production mode, we will deploy standalone executable files without the source code.

1. Run the `judgels dist <app>` command. It will create a zip file JUDGELS_HOME/dist/**<app>-<version>.zip**.
2. Copy the zip file to the target machine, in its own JUDGELS_HOME/dist directory.
3. Unzip the file. There should be a directory JUDGELS_HOME/dist/<app>-<version>.
4. If you run in HTTPS, you have to create a directory **conf** inside the above directory. This is probably a Play framework bug, and has been reported in this GitHub issue.
5. Run the `judgels start <app> <version>` or `judgels start-https <app> <version>` command.

Setting Nginx reverse proxy

The URLs like http://localhost:900x are ugly. We can set up nice domain names using Nginx reverse proxy.

1. Install Nginx.
2. Set up virtual hosts. Assume that we want to set up Jophiel. Create a file named **jophiel** in **/etc/nginx/sites-available/** with this content:

   ```
   server {
       listen 80;
       server_name jophiel.judgels.local;

       location / {
           proxy_pass                http://localhost:9001;
           proxy_set_header          Host $host;
           proxy_set_header          X-Real-IP $remote_addr;
           proxy_set_header          X-Forwarded-For $proxy_add_x_
    ↪forwarded_for;
           proxy_connect_timeout     150;
           proxy_send_timeout        100;
           proxy_read_timeout        100;
       }
   }
   ```

   The virtual host setting files for the other applications are similar. Just modify the server name and port number accordingly. The above server name is recommended for local development setup.
3. Enable the virtual host.

   ```
   cd /etc/nginx/sites-enabled
   sudo ln -s ../sites-available/jophiel .
   ```

4. Reload Nginx.
5. Make the server name point to the server IP address. For local development setup, this can be done by adding this line to **/etc/hosts**:

   ```
   127.0.0.1    jophiel.judgels.local
   ```

   For production setup, add the subdomain on your domain management web interface.
6. That's it. The Judgels application can be opened on your browser using the new server name (in this case, http://jophiel.judgels.local).

Command-line tool

Judgels comes with a handy command-line tool to make repetitive tasks easier.

Installation

First, make sure that you:

- Have a directory that will hosts all Judgels repositories. This directory is called Judgels base directory.
- Have cloned **judgels** repository to the base directory.
- Have Python 3 installed.

Open your **~/.bashrc** (or create one), and add the following lines.

```
export JUDGELS_HOME=<your-Judgels-home>
alias judgels="python3 JUDGELS_HOME/judgels/scripts/terminal.py"
```

Restart your terminal to activate the script.

Usage

All commands take the form of `judgels <command> [ <args> ... ]`. In all commands, <*repo*> is one of Judgels' repositories, while <*app*> is one of Judgels' applications, in lowercase (like jophiel, sandalphon, etc.). Omit `judgels-` prefix for <*repo*> and <*app*>. For example, to clean the build in **judgels-play-commons** repository, run `judgels clean play-commons`.

The available commands are as follows.

judgels clean <repo> Cleans the build in <repo>. This is equivalent to running `./activator clean` in <repo>.

judgels dist <repo> Creates a standalone distribution for <repo>. This is equivalent to running `./activator clean && ./activator dist` in <repo>.

judgels kill <app> Kills a running Judgels application <app> in start mode (i.e., that was run by `judgels start <app>` command).

judgels pull <repo> Pulls changes from the repository <repo> and all its dependencies, using `--rebase` strategy. If any of the repository or its dependencies is not present, it will be cloned.

<repo> can be `--all`; this will pull/clone all Judgels repositories.

judgels push <repo> Pushes changes from the repository <repo> and all its dependencies.

<repo> can be `--all`; this will push all Judgels repositories.

judgels release <version> Bumps a new version for all Judgels repositories. For Judgels admins only.

judgels run <app> [<port>] Runs the application <app>. This is equivalent to running `./activator run [ -Dhttp.port=<port> ]` in <repo>.

If <port> is omitted, the defaults are:

- jophiel: 9001
- sandalphon: 9002
- sealtiel: 9003

- uriel: 9004
- michael: 9005
- jerahmeel: 9006

judgels start <app> <version> [<port>] Starts the application from the standalone distribution created using `judgels dist` command. This is equivalent to running the standalone executable on the specified port. The Judgels application version must be specified. The default ports are the same as above.

judgels start-https <app> <version> [<port>] Same as above, but using HTTPS.

judgels status For each Judgels application, it will be output to the screen whether or not is currently running in start mode.

Jophiel (Single Sign-On)

In this section, you will learn how to:

- Install Jophiel.
- Configure Jophiel.
- Grant administrator access to users.
- Add Jophiel clients.
- Use additional features for administrators.

Table of Contents

Setup

Installing Jophiel

First, follow the *main Judgels setup* instructions if you haven't. This means that you should have installed Jophiel by running

```
judgels pull jophiel
```

and should have modified the common configuration keys in **conf/application.conf** and **conf/db.conf**.

Configuring Jophiel

This section will cover the specific configuration keys for Jophiel.

play.mailer.{host, port, ssl, user, password} SMTP credentials configuration for sending user account related emails.

jophiel.idToken.key.private An RSA private key for generating ID token required for OpenID Connect protocol. You can generate one using `ssh-keygen` command. Make sure to select RSA as the algorithm.

jophiel.client.{labels, targets} The list of public Judgels Play applications you want to inform to the users on the welcome page. This is useful in order to users that have just logged in not to get lost in SSO. Put the list of URLs in **targets**, and the corresponding link labels in **labels**.

noreply.{name, email} The name and email of "noreply" user for sending user account related emails.

aws.avatar.s3.use Whether to use AWS as the storage for user avatars. If set to true, then the rest of the **aws.avatar.*** keys below should be modified accordingly.

aws.avatar.s3.{bucket.name, bucket.regionId} The bucket name and bucket region ID for the S3 storage.

aws.avatar.cloudFront.baseUrl The base URL for CloudFront for the S3 storage. You must set up CloudFront.

aws.key.use Whether a pair of keys must be used for connecting to S3. For example, if the EC2 that hosts Jophiel has been associated to a role that has permission for connecting to S3, then this value should be false.

aws.key.{access, secret} If a pair of keys must be used, then these are the access and secret keys, respectively.

recaptcha.registration.use Whether to use reCAPTCHA for the registration form.

recaptcha.registration.key.{site, secret} If reCAPTCHA is used, then these are the site and secret keys, respectively.

Running Jophiel

See *Running Judgels Play applications*.

Adding initial admin

Jophiel has been successfully installed and configuring. Now, we need to have a user with admin role.

1. Open Jophiel. You will be presented with a login screen. Choose Register.
2. Register a user to be assigned as admin.
3. Validate the user by following the validation email. If you don't have a valid SMTP server setup, you can do the validation manually by setting the **emailVerified** column to **1** in the corresponding row in the **jophiel_user_email** table.
4. Manually assign this user as admin, by setting the **roles** column to **user,admin** in the corresponding row in the **jophiel_user** table.
5. Log in to Jophiel. Verify that you can view the **Users** menu on the left.

Manual

Adding clients

A Jophiel client is an application that uses Jophiel for authentication and authorization. To add a Jophiel client, perform these steps.

1. Open Jophiel and click **Clients** menu on the left.
2. Click **Create New**.
3. Fill in these values:

   Client Name The client name. For example: **Sandalphon #1**.

   Application Type Choose **Web Server**.

   Redirect URIs Fill **<client base URL>/verify**. For example: **http://localhost:9002/verify**.

   Scopes For the current version, choose **OPENID** and **OFFLINE_ACCESS**.

4. Click **Create New**.

A Jophiel client has been successfully created with the corresponding client JID and client secret. The client can then connect to Jophiel using the JID and secret.

Managing users

You can create and edit users manually in the **Users** menu. Also, you can view the list of unverified users, and you can also resend the verification emails.

View applications as another user

In each Judgels Play application that uses Jophiel, you can use the application as another user. Simply enter the username in the **Viewpoint** widget on the sidebar, and click **View As**. If you are finished, you can reset the view to yourself again.

This is useful for testing and investigating.

Managing activities

You can view all activities of all users in all Jophiel clients in the **Activities** menu on the sidebar.

Managing autosuggestion items

When a user is editing their profile, they will fill in institution, city, and province. The values can be autosuggested from existing values. You can view all values in **Autosuggestions** menu in the sidebar. In the future, duplicate values should be able to be merged.

Sealtiel (Message Gate)

In this section, you will learn how to:

- Install Sealtiel.
- Install the message queue service (RabbitMQ).
- Configure Sealtiel.
- Adding Sealtiel clients.
- Watching messages.

Table of Contents

Setup

Installing RabbitMQ

Sealtiel internally uses **RabbitMQ**. So, it must be installed first. Follow the installation instruction here: Downloading and Installing RabbitMQ. Make sure you install at least version **3.5.x**.

Starting RabbitMQ

Run

```
sudo rabbitmq-server
```

If RabbitMQ was installed correctly, this will be output:

```
                RabbitMQ 3.5.X. Copyright (C) 2007-2014 GoPivotal,␣
↪Inc.
##  ##      Licensed under the MPL.  See http://www.rabbitmq.com/
##  ##
##########  Logs: /usr/local/var/log/rabbitmq/rabbit@localhost.log
######  ##        /usr/local/var/log/rabbitmq/rabbit@localhost-sasl.
↪log
##########
            Starting broker... completed with X plugins.
```

Enabling management plugin

After you installed RabbitMQ, enable the management plugin:

```
sudo rabbitmq-plugins enable rabbitmq_management
```

If everything goes well, you should be able to open the management web interface on http://localhost:15672. You can login with the default user (username: **guest**, password: **guest**).

Configuring RabbitMQ

For security, you must change the guest user password.

1. Log in to the management web interface.
2. Click **Admin** tab.
3. Click **guest** user.
4. On the **Update this user** section, enter new password.
5. Click **Update user**.

Installing Sealtiel

First, follow the *main Judgels setup* instructions if you haven't. This means that you should have installed Sealtiel by running

```
judgels pull sealtiel
```

and should have modified the common configuration keys in **conf/application.conf** and **conf/db.conf**.

Configuring Sealtiel

This section will cover the specific configuration keys for Sealtiel.

sealtiel.{username, password} The credentials for logging in to Sealtiel. Note that Sealtiel does not use Jophiel for logging in.

rabbitmq.{host, port, username, password, virtualHost} The credentials for connecting to RabbitMQ. In most cases, you only need to change **rabbitmq.password** to the new guest password.

Running Sealtiel

See *Running Judgels Play applications*.

To check whether Sealtiel can connect to RabbitMQ, choose the **Connection** menu in Sealtiel's left sidebar. The connection status will be shown.

Manual

Adding Sealtiel clients

A Sealtiel client is an application that uses Sealtiel as a bridge for sending and receiving asynchronous messages. Since Sealtiel is currently only used for sending and receiving grading requests and responses, there are only two kinds of applications that use Sealtiel: Gabriel and the applications that can request grading (Sandalphon, Uriel, and Jerahmeel).

To add a Sealtiel client, perform these steps.

1. Open Sealtiel and click **Clients** menu on the left.
2. Click **Create New**.
3. Fill in these values:

   Name The client name. For example: **Uriel #2**.
4. Click **Create New**.

A Sealtiel client has been successfully created with the corresponding client JID and client secret. The client can then connect to Sealtiel using the JID and secret.

Adding acquaintances

For client A to be able to send message to client B, client B must registered as client A's "**acquaintance**". Think of acquaintance as a directed edge between clients.

To add acquaintances to a Sealtiel client, perform these steps.

1. Open Sealtiel and click **Clients** menu on the left.
2. Click the Enter icon on the rightmost column of the corresponding client row.
3. Add the client name(s) as a new acquaintance.

For example, when setting up Sandalphon and Gabriel to work together, you must:

- Add Sandalphon and Gabriel as Sealtiel clients.
- Add Gabriel as Sandalphon's acquaintance.
- Add Sandalphon as Gabriel's acquaintance.

Watching messages

You can watch the message flow between clients in the **Queues** tab on the management web interface.

Sandalphon (Repository Gate)

In this section, you will learn how to:

- Install Sandalphon.

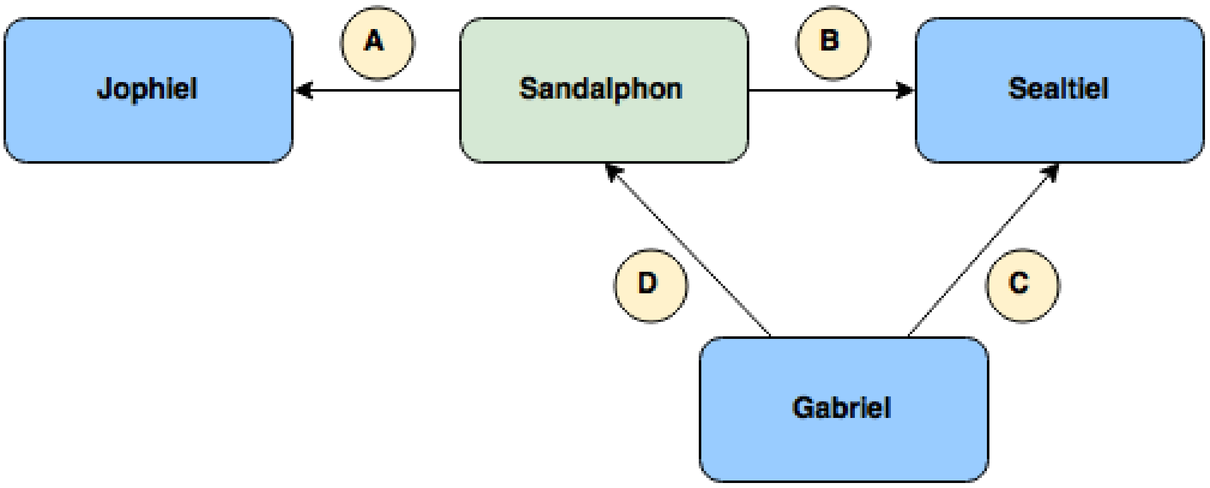
- Configure Sandalphon.
- Grant administrator access.
- Adding Sandalphon clients and graders.

Table of Contents

Setup

Installing dependencies

Sandalphon depends on other Judgels applications to run correctly. Here is the dependency diagram.



An arrow pointing from A to B means that A depends on B. The dependencies between applications are described as follows.

1. Sandalphon connects to Jophiel for user authentication and authorization.
2. Sandalphon connects to Sealtiel for sending grading requests and polling grading responses.
3. Gabriel connects to Sealtiel for polling grading requests and sending grading responses.
4. Gabriel connects to Sandalphon for fetching test cases.

Installing Sandalphon

First, follow the *main Judgels setup* instructions if you haven't. This means that you should have installed Sandalphon by running

```
judgels pull sandalphon
```

and should have modified the common configuration keys in **conf/application.conf** and **conf/db.conf**.

Configuring Sandalphon

First, let's configure the dependencies so that they can work with Sandalphon. During the configuration, we will set some configuration keys in Sandalphon's **conf/application.conf**.

1. Add Sandalphon as a client in Jophiel. Then, assign the client JID and secret values to Sandalphon's **jophiel.clientJid** and **jophiel.clientSecret**. Assign **jophiel.baseUrl** to Jophiel's base URL.
2. Add Sandalphon as a client in Sealtiel. Then, assign the client JID and secret values to Sandalphon's **sealtiel.clientJid** and **sealtiel.clientSecret**. Assign **sealtiel.baseUrl** to Sealtiel's base URL.
3. Add Gabriel as a client in Sealtiel. Then, assign the client JID and secret values to **Gabriel**'s **sealtiel.clientJid** and **sealtiel.clientSecret**. Assign **Gabriel**'s **sealtiel.baseUrl** to Sealtiel's base URL. Finally, assign the client JID to **Sandalphon**'s **sealtiel.gabrielClientJid**.
4. Add acquaintances between Sandalphon and Gabriel, vice-versa, in Sealtiel.

Running Sandalphon

See *Running Judgels Play applications*.

Adding initial admin

Sandalphon has been successfully installed and configured. Now, we need to have a user with admin role.

1. Find your user JID in Jophiel.
2. Set the **roles** column to **user,admin** in the corresponding row (that contains your user JID) in the **sandalphon_user** table.
3. Re-log in to Sandalphon. Verify that you can view the **Clients** and **Graders** menu on the left.

Manual

Adding clients

A Sandalphon client is an application that uses Jophiel for authentication and authorization. To add a Jophiel client, perform these steps.

1. Open Jophiel and click **Clients** menu on the left.
2. Click **Create New**.
3. Fill in these values:

   Client Name The client name. For example: **Sandalphon #1**.

Application Type Choose **Web Server**.

Redirect URIs Fill **<client base URL>/verify**. For example: **http://localhost:9002/verify**.

Scopes For the current version, choose **OPENID** and **OFFLINE_ACCESS**.

4. Click **Create New**.

A Sandalphon client has been successfully created with the corresponding client JID and client secret. The client can then connect to Sandalphon using the JID and secret.

Adding Gabriel

Finally, add Gabriel as a Sandalphon's grader. This is required to allow Gabriel to fetch test cases from Sandalphon. It is worth noting that Gabriel connects directly to Sandalphon for fetching test cases, not via Sealtiel, since this should be a synchronous operation.

1. Open Sandalphon and click **Graders** menu on the left.
2. Click **Create New**.
3. Fill in these values:

   Name The grader name. For example: **Gabriel #1**.

4. Click **Create New**.

Assign the JID and secret you obtained to Gabriel's **sandalphon.clientJid** and **sandalphon.clientSecret**. Assign Gabriel's **sandalphon.baseUrl** to Sandalphon's base URL.

Uriel (Competition Gate)

In this section, you will learn how to:

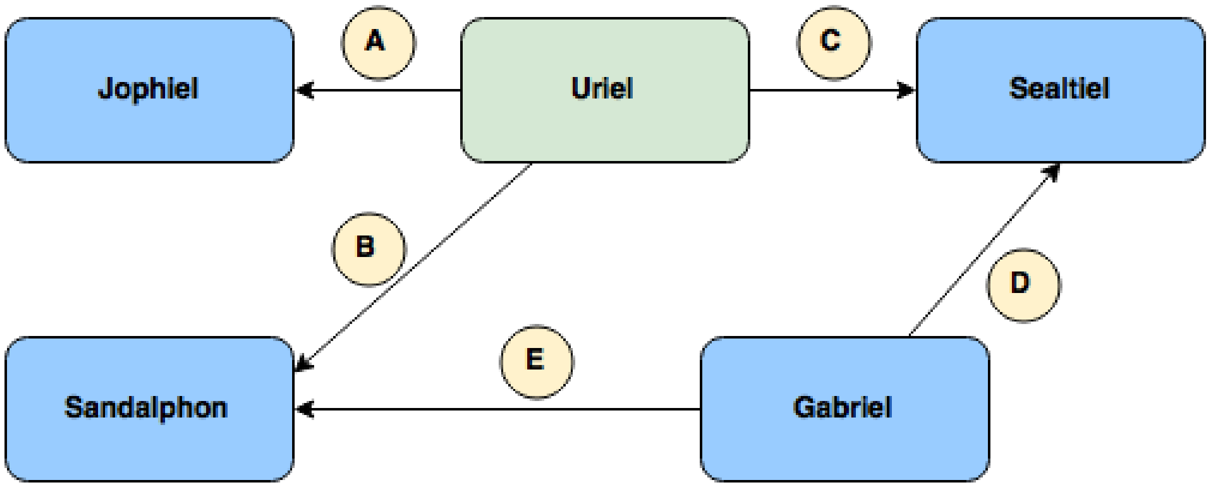
- Install Uriel.
- Configure Uriel.
- Grant administrator access.

Table of Contents

Setup

Installing dependencies

Uriel depends on other Judgels applications to run correctly. Here is the dependency diagram.



An arrow pointing from A to B means that A depends on B. The dependencies between applications are described as follows.

1. Uriel connects to Jophiel for user authentication and authorization.
2. Uriel connects to Sandalphon for fetching resources (problems/lessons).
3. Uriel connects to Sealtiel for sending grading requests and polling grading responses.
4. Gabriel connects to Sealtiel for polling grading requests and sending grading responses.
5. Gabriel connects to Sandalphon for fetching test cases.

Note that the Gabriel used for grading submissions from Uriel can be the same Gabriel used for grading submissions from Sandalphon.

Installing Uriel

First, follow the *main Judgels setup* instructions if you haven't. This means that you should have installed Uriel by running

```
judgels pull uriel
```

and should have modified the common configuration keys in **conf/application.conf** and **conf/db.conf**.

Configuring Uriel

First, let's configure the dependencies so that they can work with Uriel. During the configuration, we will set some configuration keys in Uriel's **conf/application.conf**.

1. Add Uriel as a client in Jophiel. Then, assign the client JID and secret values to Uriel's **jophiel.clientJid** and **jophiel.clientSecret**. Assign **jophiel.baseUrl** to Jophiel's base URL.
2. Add Uriel as a client in Sealtiel. Then, assign the client JID and secret values to Uriel's **sealtiel.clientJid** and **sealtiel.clientSecret**. Assign **sealtiel.baseUrl** to Sealtiel's base URL.

3. Add Uriel as a client in Sandalphon. Then, assign the client JID and secret values to Uriel's **sandalphon.clientJid** and **sandalphon.clientSecret**. Assign **sandalphon.baseUrl** to Sandalphon's base URL.
4. Add Gabriel as a client in Sealtiel. Then, assign the client JID and secret values to **Gabriel**'s **sealtiel.clientJid** and **sealtiel.clientSecret**. Assign **Gabriel**'s **sealtiel.baseUrl** to Sealtiel's base URL. Finally, assign the client JID to **Uriel**'s **sealtiel.gabrielClientJid**.
5. Add acquaintances between Uriel and Gabriel, vice-versa, in Sealtiel.

The rest of configuration keys are:

aws.{teamAvatar, submission, file}.s3.use Whether to use AWS as the storage for {team avatars, submission files, contest files}. If set to true, then the rest of the **aws.{teamAvatar, submission, file}.*** keys below should be modified accordingly.

aws.{teamAvatar, submission, file}.s3.bucket.name The bucket name for the S3 storage.

aws.{teamAvatar, submission, file}.s3.bucket.regionId The bucket region ID for the S3 storage. If not present, **aws.global.s3.bucket.regionId** will be used.

aws.teamAvatar.cloudFront.baseUrl The base URL for CloudFront for the S3 storage. You must set up CloudFront.

aws.{teamAvatar, submission, file}.key.use Whether a pair of keys must be used for connecting to S3. For example, if the EC2 that hosts Jophiel has been associated to a role that has permission for connecting to S3, then this value should be false. If not present, then **aws.global.key.use** will be used.

aws.{teamAvatar, submission, file}.key.{access, secret} If a pair of keys must be used, then these are the access and secret keys, respectively. If not present, then **aws.global.key.{access, secret}** will be used.

Running Uriel

See *Running Judgels Play applications*.

Adding initial admin

Uriel has been successfully installed and configured. Now, we need to have a user with admin role.

1. Find your user JID in Jophiel.
2. Set the **roles** column to **user,admin** in the corresponding row (that contains your user JID) in the **uriel_user** table.
3. Re-log in to Uriel. Verify that you can view the **Users** menu on the left.

Gabriel (Grader)

In this section, you will learn how to:

- Install Gabriel.
- Install the sandbox engine (Moe).
- Configure Gabriel.
- Running the graders.

Table of Contents

Setup

Requirements

Internally, Gabriel runs the contestants' programs in a sandbox: Moe Contest Environment.

The following are necessary for install Gabriel and the Moe sandbox correctly.

- Linux. We have only tested Gabriel in Ubuntu 14.04, so we recommend you to use it. Note that Moe cannot be installed in Windows or OS X.
- GCC.

Currently, programming languages that are supported are hardcoded to Gabriel: C, C++, Pascal, and Python 3. The following are necessary in order to use the languages:

- GCC >= 4.7 (for C/C++ language support)
- Free Pascal (for Pascal language support)
- Python >= 3 (for Python 3 language support)

Installing Gabriel

First, follow the *main Judgels setup* if you haven't. Then, install Gabriel:

```
judgels pull gabriel
```

This will clone Gabriel and Moe repositories to your Judgels base directory. Next, we will need to build the Moe sandbox program.

Note: The next three sections (Moe, control groups, quota) can be skipped if you plan to install Moe later. Just comment out the keys **moe.{isolatePath, iwrapperPath}** in the configuration. Gabriel can still run but the contestants' programs will not be sandboxed (which is dangerous).

This can be useful if you want to test Gabriel's connection with the other applications first.

Building Moe

Run these commands inside **moe** repository.

```
./configure
make
```

If the above commands finished correctly, then you will have two executable files:

- Isolate (**obj/isolate/isolate**): the main sandbox
- Interactive wrapper (**obj/eval/iwrapper**): wrapper for interactive problems

The above executables are necessary for Gabriel. If you cannot find them, that means your build failed. Resolve the errors and try again.

Installing control groups in Linux

Isolate needs control groups feature in Linux for sandboxing contestants' programs. You need to install it:

```
sudo apt-get install cgroup-bin
```

Then, we have to enable the memory and swap accounting in control groups. Follow these steps.

1. Add swap partition to your system if it does not have any. Note that a swap partition is **mandatory** for Isolate to function properly.
2. Open the **/etc/default/grub** using sudo privilege.
3. Modify the line containing **GRUB_CMDLINE_LINUX** as follows:

   ```
   GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"
   ```

4. Update the GRUB:

   ```
   sudo update-grub
   ```

5. Reboot the machine.
6. Verify that either the **/sys/fs/cgroup/memory/memory.memsw.limit_in_bytes** file or **/sys/fs/cgroup/memory/memory.soft_limit_in_bytes** file exists.

Enabling quota support in Linux

Quota support must be enabled so that Isolate can limit the disk usage of contestants' programs.

1. Install the kernel that support quota.

   ```
   sudo apt-get install linux-image-extra-virtual
   ```

If prompted, choose "keep the local version currently installed".

2. Edit **/etc/fstab** using sudo. Iinclude **usrquota** and **grpquota** in the desired partition:

```
LABEL=cloudimg-rootfs   /     ext4   defaults,usrquota,grpquota ␣
↪0 0
```

3. Reboot the machine.
4. Enable quota modules:

```
sudo depmod -a
sudo modprobe quota_v1
sudo modprobe quota_v2
sudo echo quota_v1 >> /etc/modules
sudo echo quota_v2 >> /etc/modules
```

5. Install quota package:

```
sudo apt-get install quota
```

6. Verify that quota support has been enabled. Go to **moe** directory and run:

```
obj/isolate/isolate -b1 -q50000,50 -vvv --init
```

This line must be output:

```
Quota: Set block quota 50000 and inode quota 50
```

Configuring Gabriel

Copy the default conf file by running this command in **gabriel** directory:

```
cp src/main/resources/conf/application_default.conf src/main/
↪resources/conf/application.conf
```

Then, fill the correct configuration values in **src/main/resources/conf/application.conf**. Some guides:

gabriel.baseDataDir The root directory for performing grading. For example: **/var/judgels/data/gabriel**.

sandalphon.{baseUrl, clientJid, clientSecret} Sandalphon's base URL and the required credentials to which this Gabriel connect for fetching test cases. This Gabriel must be registered in the Sandalphon, in **Graders** menu.

sealtiel.{baseUrl, clientJid, clientSecret} Sealtiel's base URL and the required credentials to which this Gabriel connect for fetching grading requests and sending grading results. This Gabriel must be registered in the Sealtiel as a client.

moe.{isolatePath, iwrapperPath} The absolute paths to Isolate and interactive wrapper executable files, respectively.

You can use more than one Gabriel for a single Sealtiel credentials. For example, you may want to use 5 machines each containing one Gabriel for running a contest in Uriel, to make grading process fast. Simply use identical Sealtiel configuration for all Gabriels.

Manual

Running grader

After the installation and configuration are finished, Gabriel can be run using Activator. Run this command in **gabriel** directory:

```
./activator
```

Then, in the Activator console, run:

```
run X
```

where X is the desired number of threads. It can be omitted if you want to use default recommended number of threads based on your processor.

CHAPTER 4

Introduction for Judgels developers

As Judgels is open source, we recommend anyone to contribute to Judgels development. This section is intended for getting new Judgels developers up to speed.

Basic concepts

This section will explain the basic concepts of Judgels codebase.

Technology stack

Language

- Java 8 for most code.
- Scala for Play's views.

Framework Play Framework 2.4.2 (Java) for Judgels Play applications.

Template engine Twirl for Judgels Play applications.

Database MySQL for Judgels Play applications.

ORM JPA (Java Persistence API) implemented with Hibernate for Judgels Play applications.

Build system SBT, wrapped in Typesafe Activator.

Sandbox Moe Contest Environment (the Isolate module) for Gabriel.

Project structure

Judgels repositories are hosted on GitHub, in Judgels organization (https://github.com/judgels).

The main repository is the **judgels** repository. The responisibilites of this repository are:

- Hosting Judgels command-line tool, which will be used for installing Judgels applications.
- Hosting Judgels documentation.
- Being the central GitHub issue tracker for all Judgels applications.

For each Judgels application <app>, there will be a repository named **<app>**, which hosts the application code. In addition, for some Judgels applications, there is **<app>-commons** repository, which contains parts of the application that can be used by other applications that depend on it:

commons Consists of basic dependencies used by all applications.

play-commons Mainly consists of:

- Base model components and their connection management with the database.
- Base view components that are shared across all Judgels Play applications. For example: base look-and-feel layout.
- Base controller components.

jophiel-commons Consists of components used by Jophiel clients for connecting to Jophiel.

sandalphon-commons Consists of components used by other Judgels applications for rendering problems and submitting solutions to Sandalphon.

gabriel-commons Consists of interfaces of grading engines.

There are also some repositories related to blackbox grading capability, which is the only grading capability right now.

gabriel-blackbox Consists of implementations of grading engines which are blackbox in nature.

sandalphon-blackbox-adapters Consists of views and forms for configuring grading in problems.

Finally there is a repository that deals with Java API between Judgels apps:

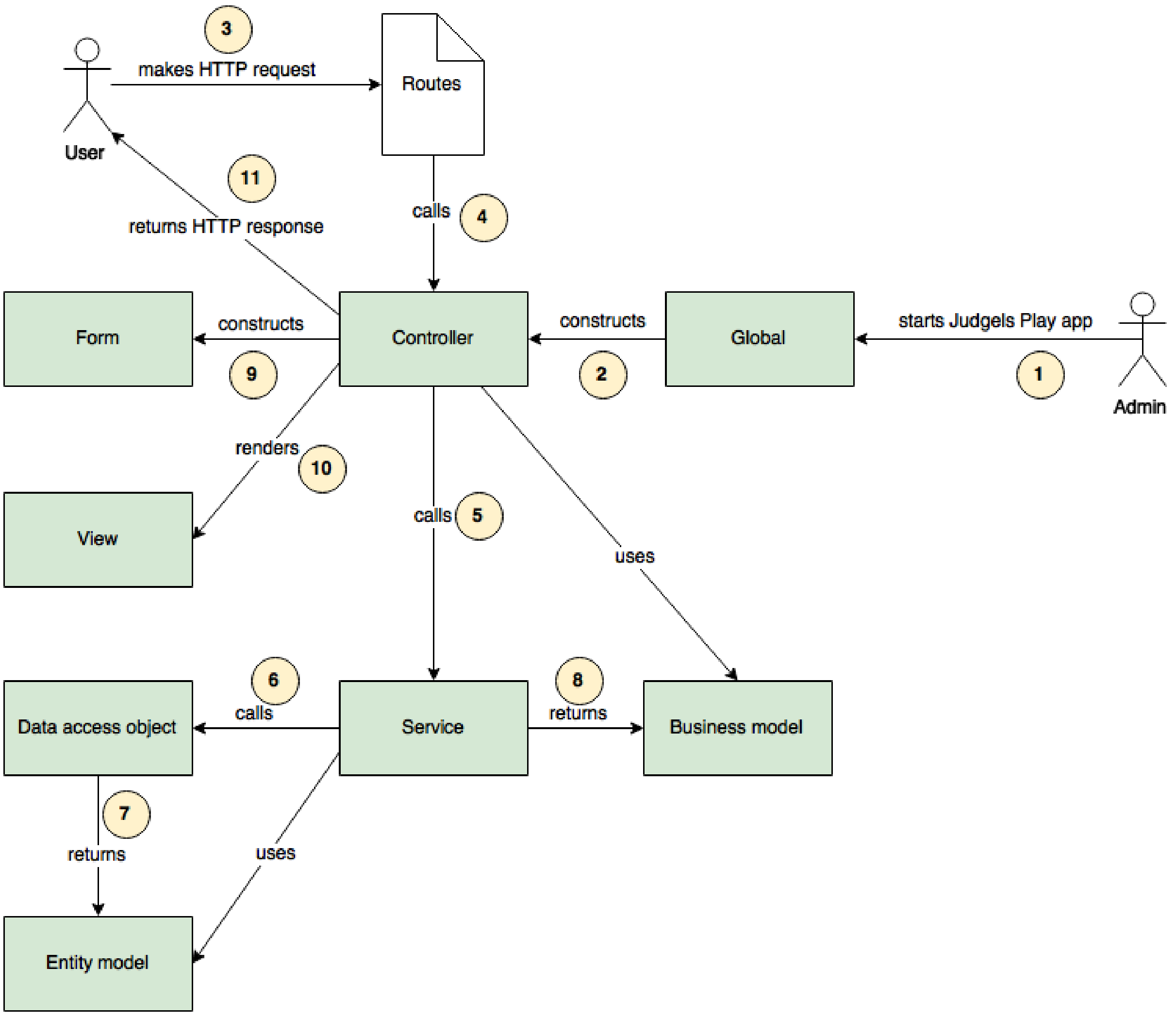
api Deals with Java API between Judgels apps.

Judgels Play application layers

Judgels Play applications are built on top of Play framework. In Play framework, the MVC (Model-View-Controller) pattern is used. In Judgels, we use two other patterns: Service and DAO (Data access object).

It will be easier to explain with a diagram. Suppose that a user wants to change his profile on Jophiel. Here is the flow of events that will happen in Jophiel, numbered from 1 through 11.

For completeness, we will explain from the start of Jophiel. Each component will be explained along with the explanation of the events.



1. Admin starts Jophiel, using `judgels start jophiel` or `judgels run jophiel` command.
2. The Play's Global object is loaded and run. It will construct all controller classes and their dependencies (services), using dependency injection with Spring Framework.
3. User makes HTTP request to change his profile, by opening the URL http://localhost:9001/profile on his browser.
4. The request is passed to the Play's Routes object. Based on the request string (**/profile**), the correct controller class and the action method is called. In this case, **UserProfileController**'s **profile()** method is called.
5. The controller checks whether the user has the right permission to perform the action. Then, controller calls the correct method in the service object responsible for user management. Services are objects that do business processes of the system. In this case, **UserProfileService**'s **updateProfile()** is called.
6. The service calls the DAO (Data Access Object) to retrieve the user entity model from the database. DAO is an object that provides interface to do queries to the database, re-

gardless of the database's SQL variants. We use Hibernate for wrapping the SQL queries. In this case, **UserDao**'s **findByJid()** is called. The user's JID is passed to the method.

7. The DAO finds the correct user record in the database, wraps it into an entity model object, and return it. Entity model is the Java class representation of a row in the database table. Fields in entity model represent columns in the database table. We use JPA (Java Persistence API) for describing the table and columns in Java. In this case, the user entity model class is **UserModel**. We also use metamodels; see Syntactically correct and type-safe JPA queries in Play 2.0 for more details.
8. The service retrieves the returned user entity model, and wraps into a user business model object. This business model represent data structure that does not depend on database, and is used by controllers, views, and the other services. They do not care how the business model was constructed. In this case, the user business model class is **User**. Normally, the business model class is similar to its entity model class counterpart. Then, the user business model object is returned by the service.
9. The controller retrieves the returned user business model. Then the controller uses the user's properties to construct the form object. In this case, the form class is **UserProfileForm**.
10. The controller passes the form to the view object to render. We use Twirl, the default Play's templating engine. The whole view and form are rendered as HTML page.
11. The controller returns an HTTP response to the user. The user is then able to fill the form for changing his profile. The flow finishes.

Database design

Judgels adapts the database design explained here: Phabricator Database Schema. Some highlights:

- Each object in Judgels has a **JID** (Judgels ID) in the form of **JID-XXX-YYYYYYYYYYYYYYYYYYYY**, where X is object type code and Y is a shortened UUID.
- No foreign keys, since we want that objects can be transferred between Judgels applications. For example, we may want to create a set of Judgels instance for OSN, and then transfer the problems back to the central repository.
- Properties that are not to be queried and have complex structure, are stored either in harddisk or in database as JSON strings.

Additionally, each object has the following fields:

- userCreate, timeCreate, ipCreate: user, time, and IP when this object is created.
- userUpdate, timeUpdate, ipUpdate: user, time, and IP when this object is updated.

Workstation setup

This section will explain how developers set up their workstation to start working on Judgels development.

Codebase setup

Basically, you have to follow all instructions on *main Judgels setup*. Make sure you are able to run/start the Judgels applications you want to develop.

In addition, you are required to download Checkstyle configuration:

```
cd $JUDGELS_HOME
judgels/scripts/download-checkstyle.sh
```

Ensure that you can run Checkstyle:

```
cd <any-judgels-repo>
activator checkstyle
```

IDE setup

Installing IDEA

As Judgels are Java projects, you can develop Judgels using either Eclipse or IntelliJ IDEA. The Judgels maintainers recommend using IntelliJ IDEA. Download the Community edition or buy the Ultimate edition here: https://www.jetbrains.com/idea/download/ and install it.

Configuring IDEA

1. Open IDEA. You will be presented with a series of dialog boxes:

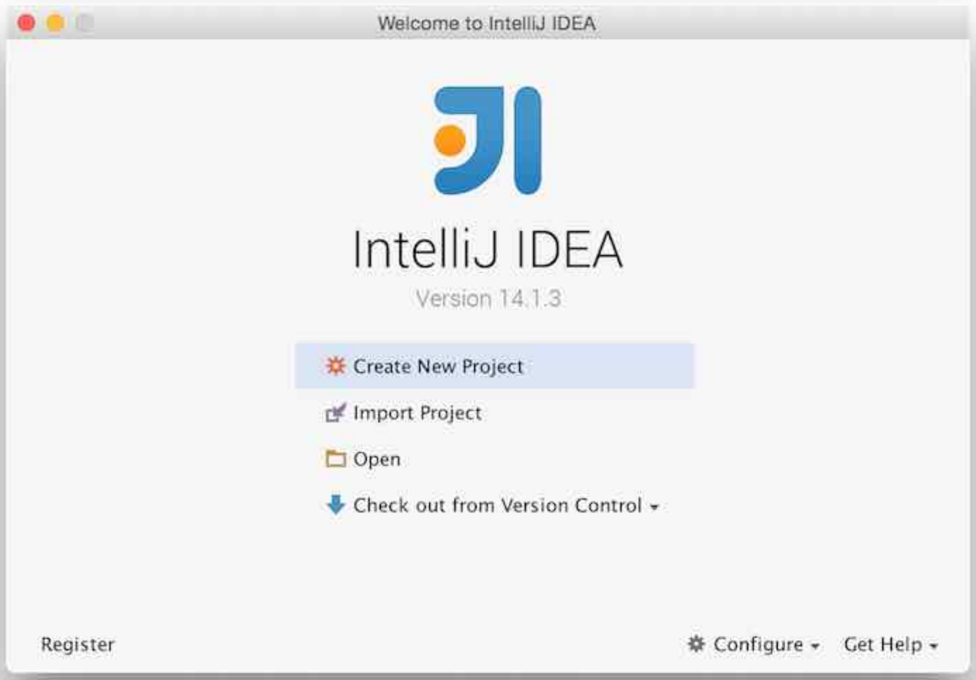
   Complete Installation Choose **I do not have a previous version of IntelliJ IDEA or I do not want to import my settings.**

   IntelliJ IDEA License Activation Enter your license key (for Ultimate edition).

   License Agreement for IntelliJ IDEA Just accept it.

   Customize IDEA Choose **Skip All and Set Defaults**.

2. The main IDEA welcome screen will appear.



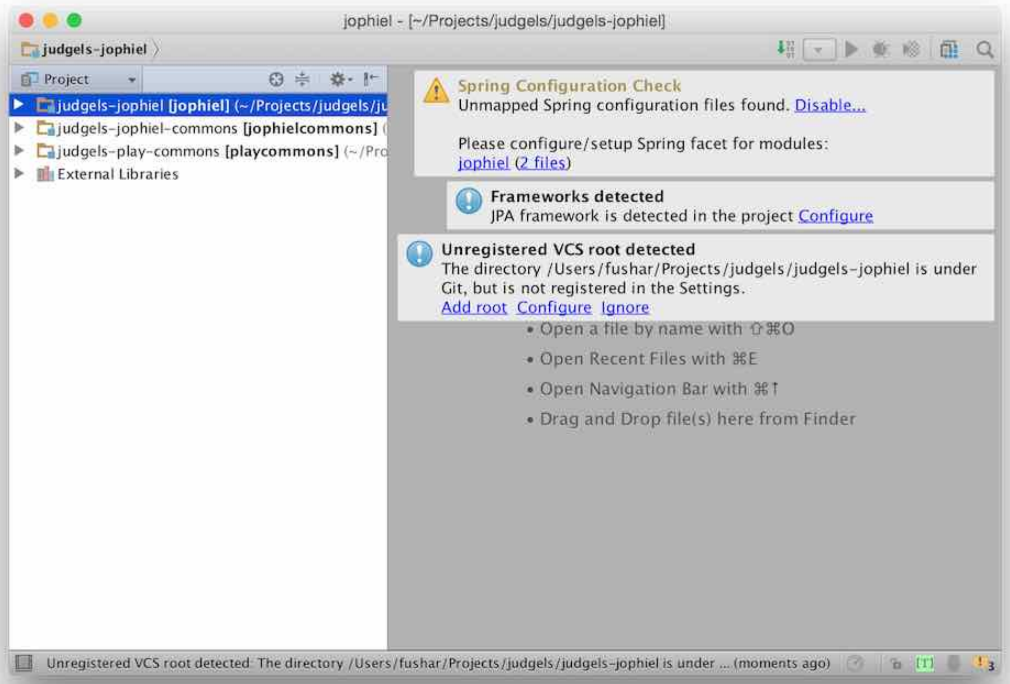
3. Click **Configure** (lower right corner) -> **Plugins**. Then, install **Scala** plugin.

IntelliJ IDEA has been successfully configured for opening Judgels projects.

Opening projects

Suppose that you want to open Jophiel project.

1. From the welcome screen, click **Open**, then select **judgels-jophiel/build.sbt** file.
2. If the **Project SDK:** dropdown is empty, click **New...** -> **JDK** beside it. Then, select your Java 8 JDK home directory.
3. Click **OK**.
4. Make sure that the Jophiel project is opened correctly. In particular, make sure that all Jophiel project dependecies are present on the projects sidebar:


jophiel - [~/Projects/judgels/judgels-jophiel]

Spring Configuration Check
Unmapped Spring configuration files found. Disable...

Please configure/setup Spring facet for modules:
jophiel (2 files)

Frameworks detected
JPA framework is detected in the project Configure

Unregistered VCS root detected
The directory /Users/fushar/Projects/judgels/judgels-jophiel is under Git, but is not registered in the Settings.
Add root Configure Ignore

Unregistered VCS root detected: The directory /Users/fushar/Projects/judgels/judgels-jophiel is under ... (moments ago)

judgels-jophiel
Project
judgels-jophiel [jophiel] (~/Projects/judgels/ju
judgels-jophiel-commons [jophielcommons]
judgels-play-commons [playcommons] (~/Pro
External Libraries
• Open a file by name with ⇧⌘O
• Open Recent Files with ⌘E
• Open Navigation Bar with ⌘↑
• Drag and Drop file(s) here from Finder

5. You will notice that there are several warning boxes on the top-right corner:

   Spring Configuration Check Ignore.

   Framework detected Click **Configure**, and choose the only **persistence.xml** file.

   Unregistered VCS root detected Click **Add root**.

   These configuration are set only in the first time you open the project.

6. You can start developing.

Fixing reverse routes object error

Soon, you will notice that Play's reverse routes object is not recognized by IDEA and will be highlighted as error. This is because the reverse routes object is a generated object. To fix the false error, right-click on these directories and choose **Mark Directory As -> Generated Sources Root**.

- judgels-jophiel/target/scala-<version>/src_managed/main
- judgels-jophiel/target/scala-<version>/twirl/main

If those directories are not present, compile/run/start the application first.

If you add new routing to the **conf/routes** file, it will not be recognized by IDEA until you compile/run/start the application (because the object has not been generated yet).

Coding style

Here are some best practices that are recommended by Judgels maintainers.

Java

1. Use 4 spaces for indentation (not tabs).
2. Use curly braces in a block even it contains only one statement.
3. Put the opening curly brace on the right of the statement:

   ```
   if (condition) {
       //
   }
   ```

4. Do not use star import (`import xxx.*;`). Import individual classes.
5. Mark classes as **final** whenever possible.
6. Use Google Guava's immutable collection whenever possible.
7. Make classes immutable whenever possible. Do not introduce setters unless really required. Initialize the class properties inside the constructor.
8. Prefer this:

   ```
   public void someFunction() {
       if (!requiredCondition1) {
           return failureMethod1();
       }

       if (!requiredCondition2) {
           return failureMethod2();
       }

       // main function code

   }
   ```

   to:

   ```
   public void someFunction() {
       if (requiredCondition1) {
           if (requiredCondition2) {

               // main function code

           } else {
               return failureMethod2();
           }
       } else {
   ```

```
        return failureMethod1();
    }
}
```

9. Put only statements that can really throws exception, inside a try-block. Pull the ones that don't outside.

Judgels documentation guide

This section is intended for developers that want to contribute in writing Judgels documentation.

Introduction

All Judgels documentation is stored in **judgels** repository, in the directory **docs**. The documentation is published on Read the Docs, here: http://judgels.readthedocs.org. The documentation will be updated every time there is a commit pushed to **judgels** repository.

The documentation is written using Sphinx. To use Sphinx, we need Python. To simplify Sphinx installation, we will use **virtualenv**.

Setup

1. Install Python.

   On Ubuntu:

   ```
   sudo apt-get install python
   ```

   On OS X (via **Homebrew**):

   ```
   brew install python
   ```

2. Install **pip**.

   On Ubuntu:

   ```
   sudo apt-get install python-pip
   ```

   On OS X: automatically installed along with Python.

3. Install **virtualenv** via **pip**:

   ```
   pip install virtualenv
   ```

4. Go to main documentation directory.

```
cd $JUDGELS_BASE_DIR/judgels/docs
```

5. Create virtual environment.

```
virtualenv env
```

6. Activate the virtual environment.

```
source env/bin/activate
```

7. Install Sphinx.

```
pip install sphinx
```

Sphinx will be ready in the directory.

Writing documentation

First, try to build the documentation. Run

```
make html
```

If everything goes well, a file **$JUDGELS_BASE_DIR/judgels/docs/_build/html/index.html** will be built. Open it on your browser to see the documentation.

The documentation is written in **reStructuredText (RST)** syntax. The root documentation source file is **docs/index.rst**. Please consult reStructuredText Primer for more details on the syntax.

Troubleshooting

If you encounter these weird errors during development:

- no value received from a submitted form even though you have filled it correctly,
- cannot bind properties to a model,

just try to clean the project (`./activator clean`).

CHAPTER 5

Overview

This is an introduction to Judgels project. This is good for getting started to know about Judgels. Please read this section first.

CHAPTER 6

Operator's Guide

This guide is for people who want to operate and manage existing Judgels applications. It covers, for example, how to:

- write problems
- set up contests
- supervise contests
- administer forums
- etc.

CHAPTER 7

Administrator's Guide

This guide is for people who want to set up working Judgels application instances. It covers, for example, how to:

- download Judgels
- install Judgels instances on one or several machines
- set up initial state of the applications
- runn necessary scripts
- etc.

CHAPTER 8

Developer's Guide

This guide is for people who want to develop Judgels. It covers, for example:

- the basic concepts of Judgels development
- how to set up the workstation for Judgels development
- the coding styles and conventions