judgels Documentation

Release 0.1.0

ia-toki

March 24, 2015

Contents

CHAPTER 1

Introduction

Judgels (**Judgment Angels**) is an open source, distributed, multi-application software for educative programming events, such as:

- competitive programming contests,
- academic programming classes/courses,
- online judge/problems archive,
- etc.

It was initiated by Ikatan Alumni Tim Olimpiade Komputer Indonesia (English: Indonesia Computing Olympiad Alumni Association). Judgels is designed to be extensible, adaptable, and modular in the hope that it can be easily modified for any educative programming related purposes in the future.

Judgels is still is being heavily developed and does not have any stable releases yet. However, anyone can view and use the codebase on GitHub at own risks.

1.1 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 five applications in Judgels:

1. Single Sign-On (**Jophiel**): authenticating and authorizing users.
2. Repository Gate (**Sandalphon**): storing programming problems and materials.
3. Competition Gate (**Uriel**): holding programming contests.
4. Middleware (**Sealtiel**): providing message queues and transmissions between applications.;
5. Grader Application (**Gabriel**): grading programming submissions.

All applications don't have to be installed in one machine. We can install an application in one machine and the others in other machines. This ensures good scalability.

We are still working on other applications for programming courses and online judges.

1.2 License

Judgels is licensed using GNU GPL version 2.

1.3 Contributions

Currently, the project is maintained by (in alphabetical order) **Ashar Fuadi** and **Jordan Fernando**. Public contributions are welcome. Please consult the Developer's Guide on how Judgels works in details.

CHAPTER 2

Setup

This page describes how to properly install Judgels on your system.

2.1 Requirements

The following packages must be installed on your machine(s), for all Judgels applications:

- Oracle Java 8
- MySQL >= 5.6
- SBT >= 0.13
- Git;
- Nginx (for reverse proxy)

All Judgels applications can run on any operating systems, except for Gabriel.

Here are the requirements for each specific Judgels application.

2.1.1 Jophiel, Sandalphon, Uriel

(no additional requirements)

2.1.2 Sealtiel

- RabbitMQ

2.1.3 Gabriel

- Linux operating system that support control groups swap limit
- Gnu Compiler Collection >= 4.7 (for C/C++ grading language support)
- Free Pascal (for Pascal grading language support)

- Python >= 3 (Python grading language support)

2.2 Installation

To install judgels, you need to clone the latest version of judgels from GitHub and run "scripts/pull-latest.sh" to clone all of its' submodules.

After cloning judgels from GitHub, you can run each applications from their directory by executing "activator" or "sbt" (only for gabriel and gabriel-commons). From the submodules, the directories with suffix "_commons" are to reduce codes duplication. Please refer to Play Framework and SBT documentation to run the applications.

2.3 Configuration

After installing judgels, you need to fill the configuration such as database, session cookie name, etc. There are default configuration files in "conf" directory or "src/resources/conf" (for Gabriel) with suffix "_default.conf". You can copy the configuration files, remove the suffix "_default" (i.e. application_default.conf -> application.conf) and configure the values accordingly.

CHAPTER 3

Single Sign On (Jophiel)

3.1 Configuration

First copy default configuration in "conf" directory: - application_default.conf -> application.conf - db_default.conf -> db.conf

Create a database for storing Jophiel data. Then change the configuration of Jophiel to point to the database in "conf/db.conf" file.

You also need to configure smtp email to support Jophiel email verification feature. You can find the configuration in "conf/application.conf" file.

Jophiel needs a directory to save data such as avatar files. You need to create one and configure the directory in "conf/application.conf" file.

Jophiel depends on judgels-play-commons to run, make sure judgels-play-commons is on the same level of directory with Jophiel.

3.2 Running Jophiel

You can run Jophiel by using "activator start" command. By default, it will listen on port 9000. You can access it via web browser using url "http://localhost:9000".

At the first run, you need to register and verify your email in order to login. After your first user is created, you can change the value of your roles in table "jophiel_user" from "user" into "user,admin" to access full feature.

3.3 Jophiel User Roles

Currently there are two roles in Jophiel:

1. Normal User

   Currently normal user can only edit their profile on Jophiel.

2. Admin

   Admin can manage (CRUD) users and manage (CRUD) clients. Jophiel clients can use Jophiel single sign on for authentication and authorization.

3.4 Single Sign On Protocol

Jophiel uses Open Id Connect Protocol for single sign on. Since Open Id Connect is built on top of OAuth 2.0 protocol, it provides authorization besides authentication. Each Jophiel Client's manage their own authentication and authorization using Jophiel's provided API, access token, and id token.

3.5 Future Target

Currently Jophiel is meant to be used for private clients of IA TOKI. In the future, we want each user to be able to use Jophiel Single Sign On to create applications.

CHAPTER 4

Programming Repository (Sandalphon)

4.1 Configuration

First copy default configuration in "conf" directory: - application_default.conf -> application.conf - db_default.conf -> db.conf

Create a database for storing Sandalphon data. Then change the configuration of Sandalphon to point to the database in "conf/db.conf" file.

Sandalphon uses Jophiel for authentication and authorization. You need to configure Jophiel's parameter in "conf/application.conf" file.

Sandalphon needs a directory to save data such as problems and submissions. You can find the configuration in "conf/application.conf" file.

Sandalphon uses Gabriel to grade programming submissions. Gabriel can only be contacted through Sealtiel that acts as queue manager and router. You need to configure the parameter in "conf/application.conf" file.

Sandalphon depends on judgels-play-commons, judgels-gabriel-commons, and judgels-frontend-commons to run, make sure judgels-play-commons, judgels-gabriel-commons, and judgels-frontend-commons are on the same level of directory with Sandalphon.

4.2 Running Sandalphon

You can run Sandalphon by using "activator start" command. By default, it will listen on port 9000. You can access it via web browser using url "http://localhost:9000".

At the first run, you need to have Jophiel's account in order to login. After you login into Sandalphon you can change the value of your roles in table "sandalphon_user_role" from "user" into "user,writer,admin" to access full feature.

4.3 Sandalphon User Roles

Currently there are three roles in Sandalphon:

1. Normal User

   Currently normal user who can do nothing on Sandalphon.

2. Writer

   Writer can manage (CRUD), test programming problems and give programming problem access to clients.

3. Admin

   Admin can manage (CRUD) clients and graders. Sandalphon's clients can render programming problem that has been given access to. Sandalphon's graders can fetch test-cases through given API.

4.4 Client's Problem Rendering

After Client registration, each Client is given client ID and client Secret for Sandalphon's API authentication. Currently, Sandalphon only has the API to render problem.

Each Problem can be given access to any client registered on Sandalphon. Client that has been given access gets "Problem Secret". Client can render problem using certain API that requires Time-based One Time Password that can be generated using "Problem Secret". TOTP is used to ensure security that programming problem can only be accessed through the link in limited duration.

4.5 Future Target

Currently Sandalphon's writer are only accessible for limited people. In the future, we want every user to become writer that can create problem and share it with each others.

CHAPTER 5

Problem Types

5.1 Blackbox Grading

In blackbox grading, user's solution is run with provided input testcase and evaluated based on the output of the solution. Blackbox grading don't evaluate the structure or style of the code. Blackbox grading also evaluate the solution performance by setting constraints such as time and memory limit.

5.2 Constraints

User solution is run in a isolated environment with some constraints. Isolation of user solution is done by using sandbox. Currently, we are using isolate as sandbox.

5.2.1 Memory Limit

The constraint memory limit restricts total memory that can be used in the solution. This means the data structure that can be created is limited to the maximum memory allowed.

5.2.2 Time Limit

The constraint time limit restricts the time user solution need to solve the problem. User's solution need to be fast enough to solve the problem within the time limit.

5.3 Evaluators

Below are the terms that are used in Judgel's evaluators:

5.3.1 Testcases

A testcase is a pair of given input and expected output based on the problem.

5.3.2 Subtasks

A subtask is a collection of testcases that are grouped by certain constraint. Each subtask is assigned to certain score. To pass a subtask, user's solution need to pass all the testcases in the subtask.

5.3.3 Test Groups

Test groups is a feature in judgels to group testcases. Sometimes, writer need to create incremental subtask where subtask 2 can only be solved if subtask 1 is solved. In order to do this, the testcases in subtask 1 must be included in subtask 2. This can be replaced by using test groups where subtask 1 contains test group 1 and subtask 2 contains test group 1 and 2.

5.3.4 Checker

In blackbox grading, sometimes there are problems with special conditions. Some of the special conditions are multiple correct answers and floating point precission. This can be solved by providing checker. A checker is a program that checks provided input, user solution's output with expected output and check if the solution is correct or not.

5.4 Problem Types

Therea re three types of programming problem:

5.4.1 1. Batch

Batch problem type require the collection of testcases. In batch problem type, the problem can be solved partially by submitting a solution that pass certain testcases.

The score is calculated by using the number of passed testcases. In total, user can get 100 score. E.g., if there is a problem with total of 20 testcases and a user's solution passed 11 testcases then the user get a total score of 55.

5.4.2 2. Batch with Subtasks

Batch with Subtasks problem type is similar to the batch problem type.

In batch with subtasks problem type, the score is calculated based on the number of passed subtasks. The total of score that a user can get is the total of score assigned

to each subtasks. E.g., if there is a problem with total of 3 subtasks with scores (20, 30, 50) and a user's solution passed subtask 1 and 2 then the user get a total score of 50.

5.4.3 3. Interactive with Subtasks

Interactive with Subtasks problem type is a interactive problem with subtask scoring. The way subtask is used is the same with batch with subtasks.

In interactive problem type, user need to submit a solution that interact with another program.

CHAPTER 6

Message Oriented Middleware (Sealtiel)

6.1 Configuration

Install RabbitMQ on your Operating System.

First copy default configuration in "conf" directory: - application_default.conf -> application.conf - db_default.conf -> db.conf

Create a database for storing Sealtiel data. Then change the configuration of Sealtiel to point to the database in "conf/db.conf" file.

Sealtiel depends on judgels-play-commons to run, make sure judgels-play-commons is on the same level of directory with Sealtiel.

6.2 Running Sealtiel

You can run Sealtiel by using "activator start" command. By default, it will listen on port 9000. You can access it via web browser using url "http://localhost:9000".

6.3 Message Queue

Message delivery in Sealtiel utilize message queue. Sealtiel uses RabbitMQ to contains the queue. In Judgels, Sealtiel acts as router to connect components with graders. Components and graders in Judgels have to be registered as client of Sealtiel to utilize the message delivery.

6.4 Polling

In message delivery, Sealtiel acts as passive components. Sealtiel only accept incoming message delivery request from Client and put it into queue. In order for Sealtiel's client to get the message, they must actively poll message from Sealtiel.

After polling the message from Sealtiel, the client need to send ack to notify if the message has been processed. If the message hasn't been processed after some times, the message will be requeued. Client can extend the acknowledgement time limit using Sealtiel's API.

6.5 Grading Distribution

Message queue in Sealtiel can be used for grading distribution purposes. Currently, each client is associated with one queue. Client can send message to each other through the queue identity. To distribute grading, all grader instance can be registered as one client and obtains one queue. All grading request message can be sent to graders' client queue. Since each grader poll from the queue every interval time, load balancer is created naturally.

6.6 Future Target

Currently all grader are registered as one client. There need to be implementation of a group of client (channel) to register each grader as one client and support more routing options.

CHAPTER 7

Grader (Gabriel)

7.1 Configuration

Install SBT on your Operating System.

First copy default configuration in "src/main/resources/conf" directory: - application_default.conf -> application.conf

Gabriel need to connect to Sealtiel to fetch grading requests. You need to configure the parameter in "src/main/resources/conf/application.conf" file.

Gabriel need to connect to Sandalphon to fetch problem evaluator files (testcases, subtask configuration, constraints, checkers, etc). You need to configure the parameter in "src/main/resources/conf/application.conf" file.

Next, you have to configure Moe. Run ./configure and make in judgels-moe repository. Gabriel uses Moe to grade submissions. You need to configure the parameter in "src/main/resources/conf/application.conf" file.

Gabriel depends on judgels-moe and judgels-gabriel-commons to run, make sure they are on the same level of directory with Gabriel.

7.2 Running Gabriel

You can run Gabriel by using "sbt" command to enter sbt console and enter "run" command in the console.

7.3 Grading Engine

Gabriel can grade types of submissions based on existing grading engines. Currently there are three grading engines as described on Problem Types:

- Batch
- Batch with Subtasks

- Interactive with Subtasks

7.4 Future Target

Currently Gabriel's grading engine are coded along with Gabriel. In the future, Gabriel's grading engine are supposed to be separated as add-on that can be fetch by Gabriel.

CHAPTER 8

Competition Gate (Uriel)

8.1 Configuration

First copy default configuration in "conf" directory: - application_default.conf -> application.conf - db_default.conf -> db.conf

Create a database for storing Uriel data. Then change the configuration of Uriel to point to the database in "conf/db.conf" file.

Uriel uses Jophiel for authentication and authorization. You need to configure Jophiel's parameter in "conf/application.conf" file.

Uriel needs a directory to save data such as team avatars. You can find the configuration in "conf/application.conf" file.

Uriel uses Sandalphon to render problems. You need to configure Sandalphon's parameter in "conf/application.conf" file.

Uriel uses Gabriel to grade programming submissions. Gabriel can only be contacted through Sealtiel that acts as queue manager and router. You need to configure the parameter in "conf/application.conf" file.

Uriel depends on judgels-play-commons, judgels-gabriel-commons, and judgels-frontend-commons to run, make sure judgels-play-commons, judgels-gabriel-commons, and judgels-frontend-commons are on the same level of directory with Uriel.

8.2 Running Uriel

You can run Uriel by using "activator start" command. By default, it will listen on port 9000. You can access it via web browser using url "http://localhost:9000".

At the first run, you need to have Jophiel's account in order to login. After you login into Uriel you can change the value of your roles in table "uriel_user_role" from "user" into "user,admin" to access full feature.

8.3 Uriel User Roles

Currently there are two main roles in Uriel:

1. Normal User

   Normal user can register to open contests, enter registered contests, solve problems, and do other contest related stuff.

3. Admin

   Admin can manage (CRUD) contests and user roles.

Besides main roles, there are also implicit roles in Uriel:

1. Coach

   Coach can start a virtual contest for the team. This role is specifically made to be used in APIO like contest.

2. Supervisor

   Supervisor can manage contest announcements, clarifications, contestants, problems, and submissions based on the access given.

3. Manager

   Manager is assigned by admin. Manager can do stuff that supervisor can do and manage supervisors and their access.

CHAPTER 9

Contest Types

Contest can be divided based on three categories:

9.1 Type

Currently, there are two contest types:

- Standard

  Standard contest are normal contest with fixed duration and user can only do contest within that duration. Standard contest have configuration of contest scoreboard freeze time. Contest scoreboard that are displayed to contestant will be freezed after the freeze time.

- Virtual

  Virtual contest consist of two kinds of duration. The first duration is the contest total duration where contest can be entered within the duration. The second one is the contest virtual duration. Once a user has entered the contest, the total time of contest is using the contest virtual duration.

  Virtual contest can be started by contestant or team coach based on the configuration.

9.2 Scope

Currently, there are two contest scopes:

- Public

  In public contest, any user can register to the contest. Public contest is also known as open contest. Public contest have configuration of register time (user can only register within this time) and max registrants (the total number of maximum user registered).

- Private

  In private contest, users are registered manually by contest supervisors or managers. Only registered user can enter the contest.

9.3 Style

Currently, there are two contest styles:

- IOI

  IOI style contest supports partial score between 0 and 100. In IOI style, the time contestant uses to solve the problem is not counted for ranking. That means there could be two or more contestants with the same rank.

- ICPC

  ICPC style contest doesn't support partial score, it only has 0 or 1 score. In ICPC style, there is a concept of penalty where a wrong answer will get time penalty. Besides wrong answer, a submitted answer by default get penalty based on the passed time since contest start. ICPC contest have configuration of the time penalty of wrong submission.

CHAPTER 10

Troubleshooting (FAQ)

10.1 Usage

10.2 Development

- A class is not found, but it exists, how?

  Try to clean the project and its dependencies.
- You encounter no value error after submitting form, what to do?

  Try to clean the project and its dependencies.
- I don't want any red underscore in my IntelliJ IDEA?

  Ashar please answer this

CHAPTER 11

Developer's Guide

To start contributing to judgels development, you can view the project hosted on github. We put the issues of all judgels applications in that repository.

You can help by submit a new issue if you found one or fix a opened issue. In order to fix the issue, you can read below for to learn of our "teachings" of judgels development.

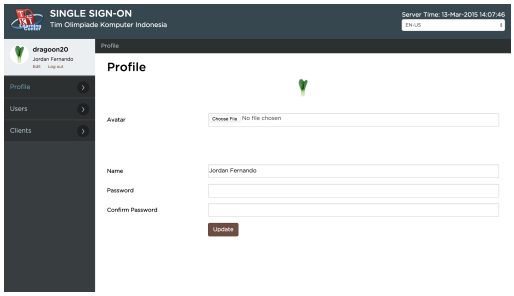
We develop judgels using Play Framework 2.3.8, SBT, and IntelliJ IDEA 14. Since, we develop many judgels application, we create the commons that can be reused within many applications.

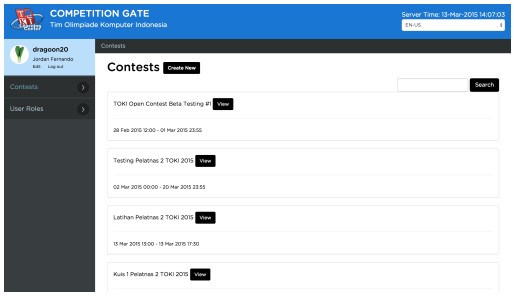
The commons are:

- Judgels Play Commons

  Judgels Play Commons consists of classes related to base layout rendering of views, less css, and javascripts. Judgels projects that use Judgels Play Commons can have the similar look and feel with every Judgels Application. The look and feel can be customized in each Judgels projects.

  Below are the images of two applications with similar look and feel using Judgels Play Commons.





- Judgels Gabriel Commons

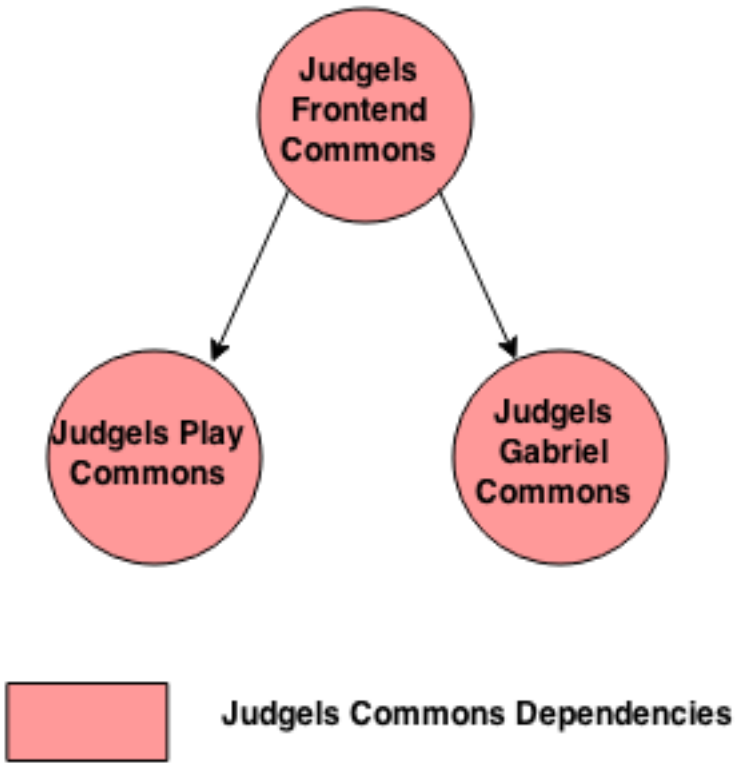
Judgels Gabriel Commons consists of classes that contains implementation of grading engines. This commons is created to separate the grading engines from Gabriel. For Gabriel, grading engines only act as the implementation and Gabriel doesn't need to know about the detail.

Judgels Gabriel Commons is also used in Judgels applications to list all available grading engines and construct grading messages that is supported by Gabriel.

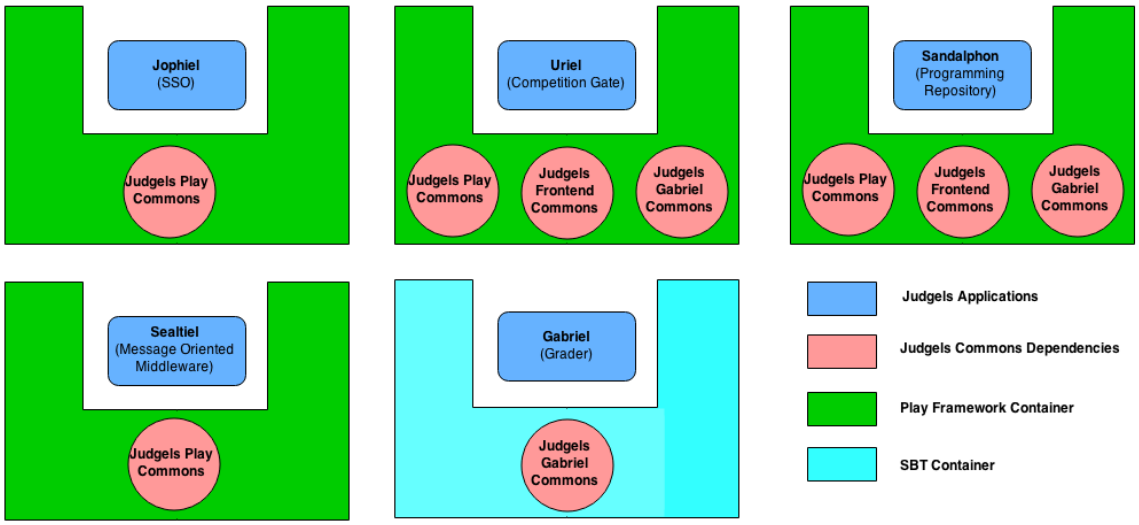
- Judgels Frontend Commons

  Judgels Frontend Commons consists of classes that helps Judgels applications to connect to each others. Some of the classes are to provide problem rendering for Sandalphon's clients, single sign on for Jophiel's clients, etc.

  Judgels Frontend Commons also depends on Judgels Play Commons for layout and Judgels Gabriel Commons to provide base submission polling class. The dependency of Judgels Frontend Commons can be seen in image below.



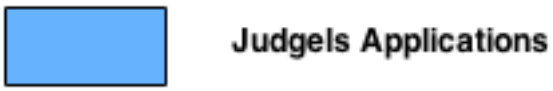
To use commons, Judgels applications directory must be on the same level as the commons directory. The build dependencies for all Judgels applications are shown in the image below.



Besides commons, Judgels applications are also connected to each other. The dependencies for Judgels applications:

- Jophiel

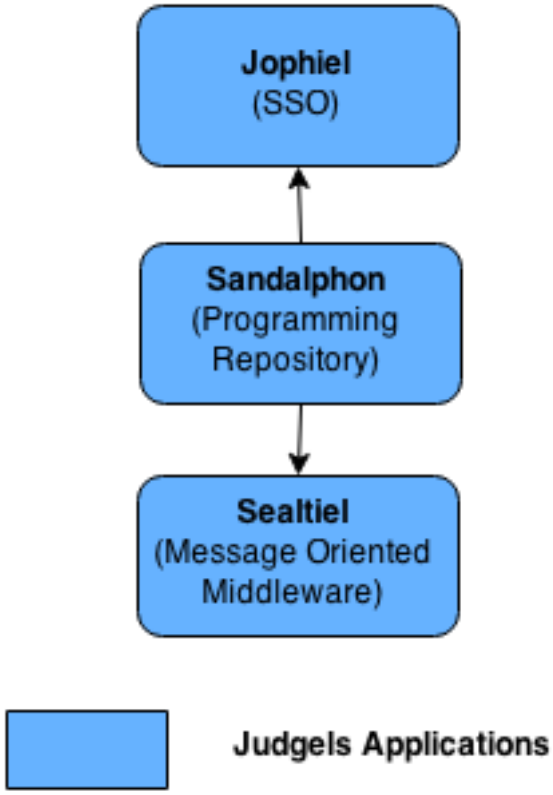
  Jophiel doesn't have any dependencies to any Judgels applications as shown in the image below. Jophiel can be run and provide single sign on service independently.



- Sandalphon

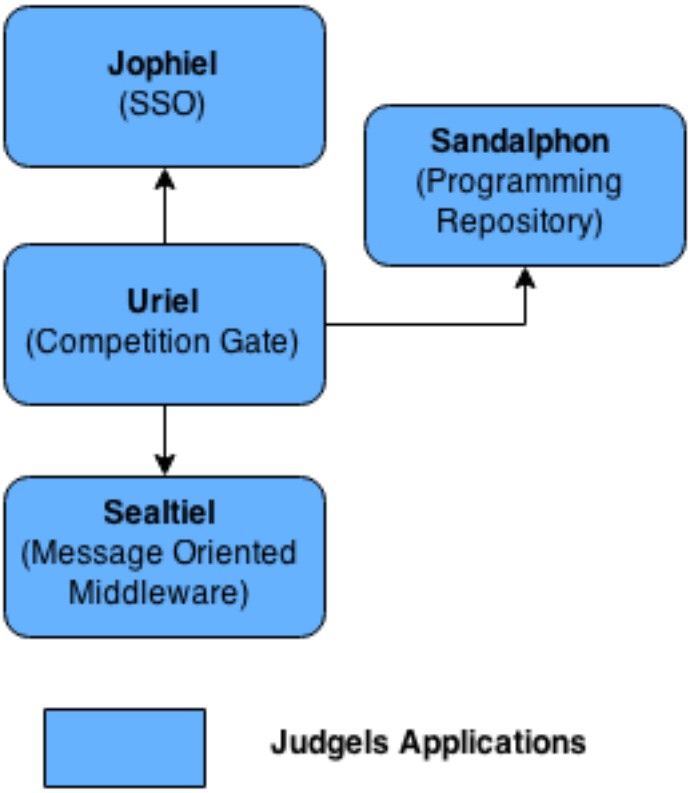
  Sandalphon has dependencies to Jophiel for authentication and authorization and to

Sealtiel for sending grading message to Gabriel.



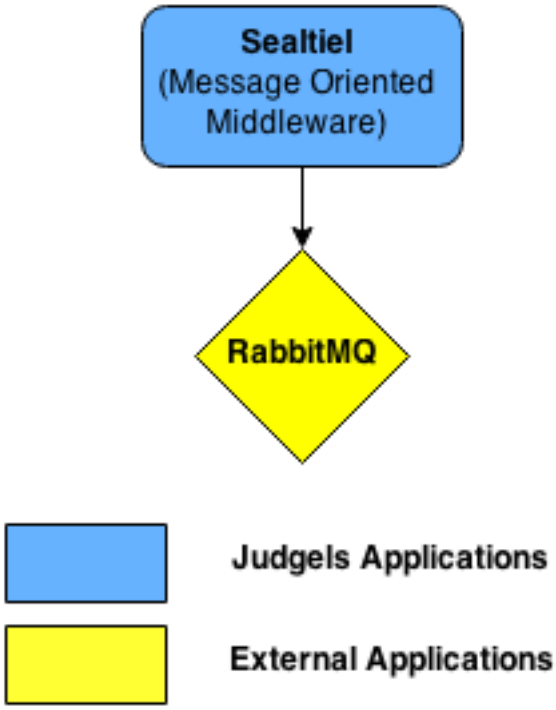
- Uriel

  Uriel has dependencies to Jophiel for authentication and authorization, to Sandalphon for rendering problems, and to Sealtiel for sending grading message to Gabriel.



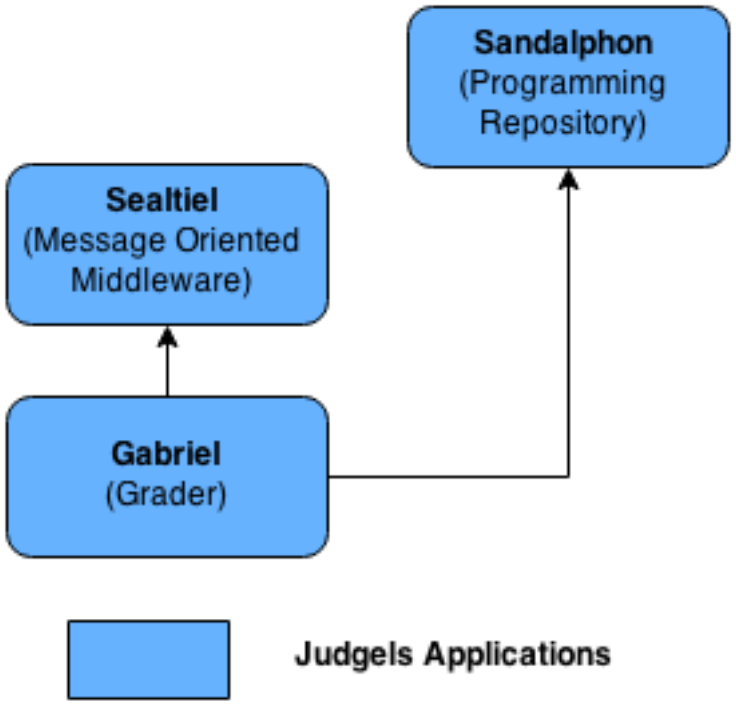
- Sealtiel

  Sealtiel has dependencies to RabbitMQ for storing messages inside queues.

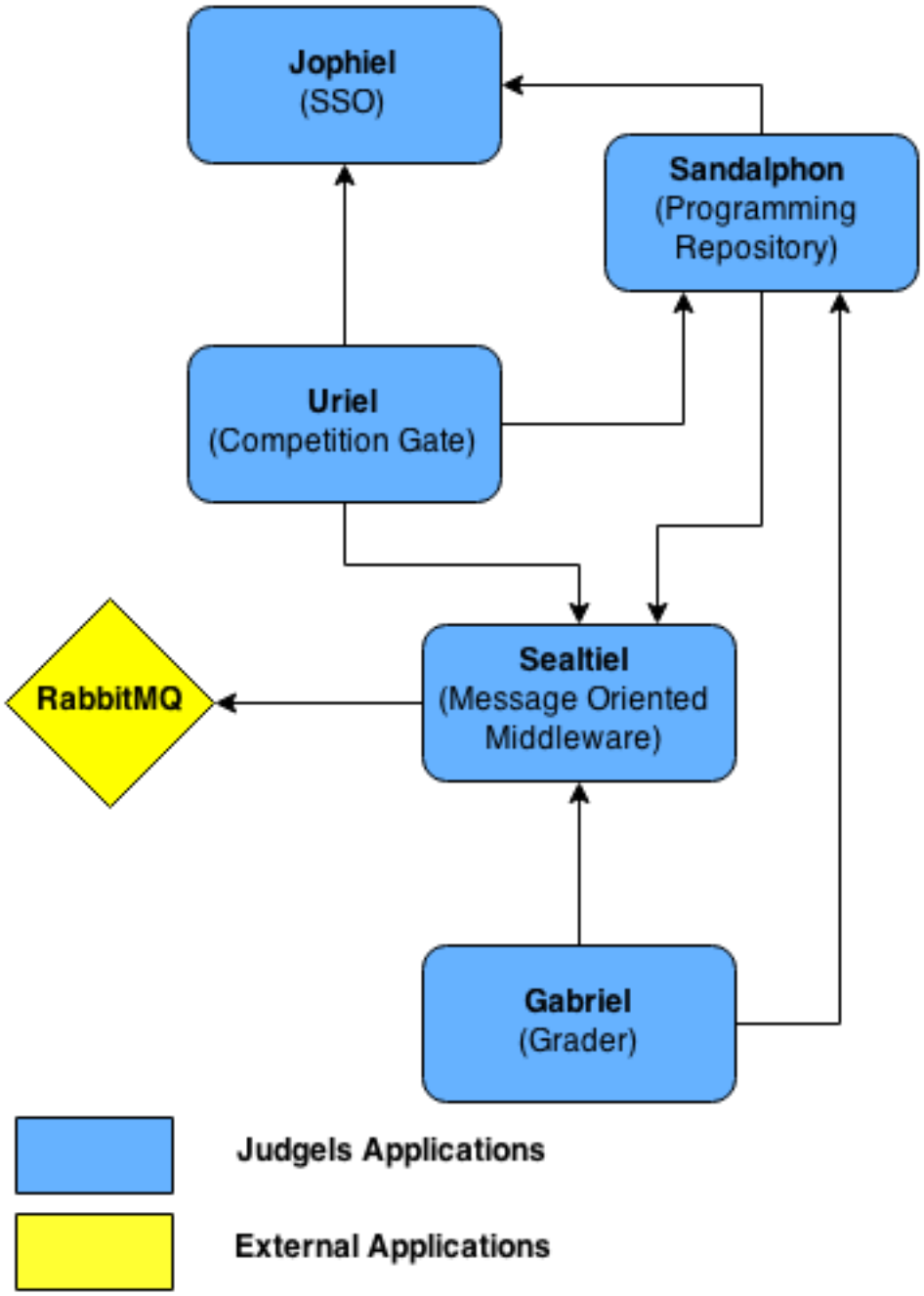


- Gabriel

  Gabriel has dependencies to Sealtiel for polling grading requests and to Sandalphon to get problem evaluator data.



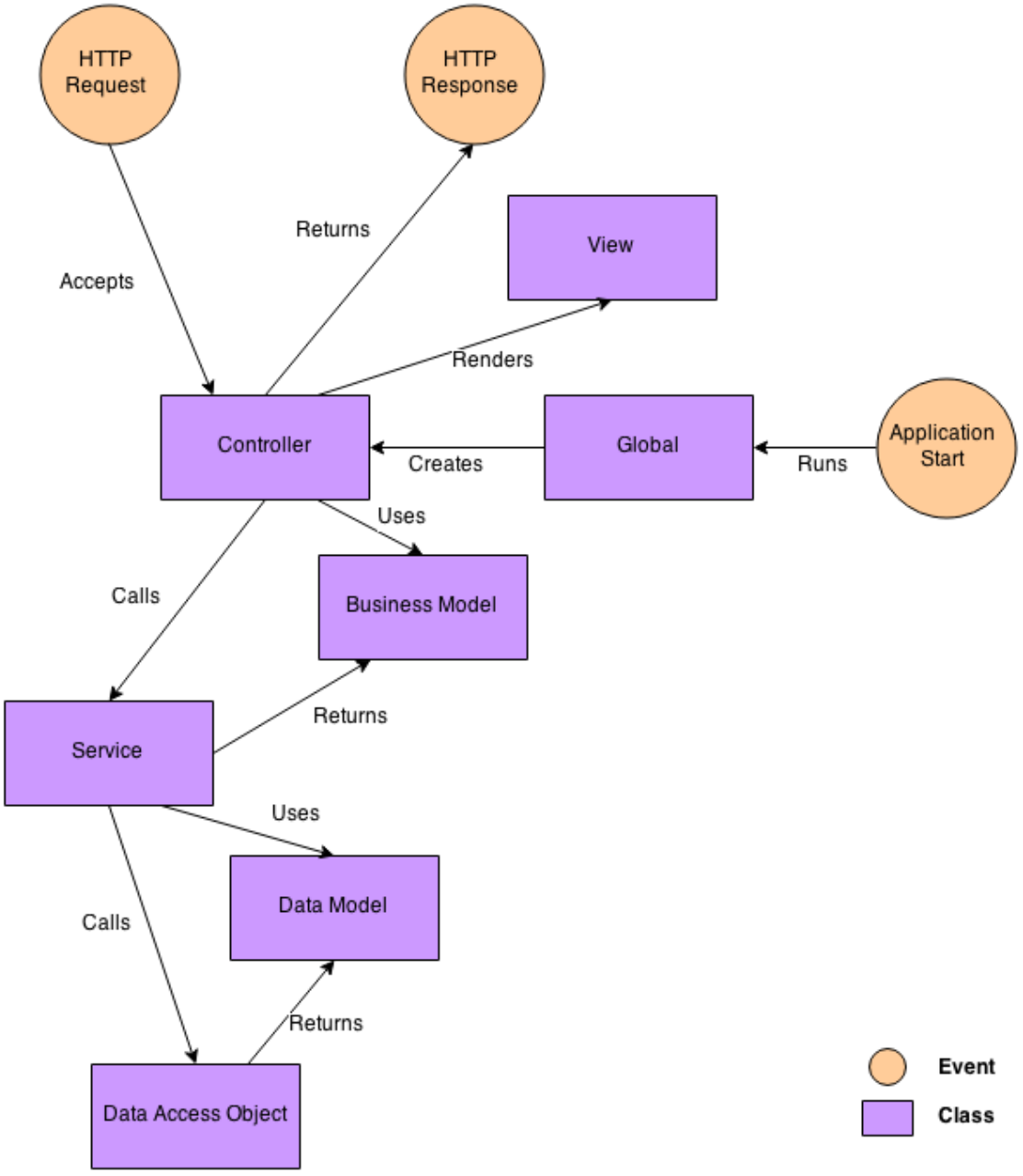
All Judgels applications dependencies can be shown in the image below.



In Judgels web applications, we use Model View Controller (MVC) architectural pattern (the same pattern with Play Framework default). We use MySQl and Hibernate ORM for the databases.

We add some improvement to the pattern by using Services and Data Access Objects. By using those patterns, we can inject dependencies to create mock objects for testing purposes in the future.

Below are the image showing the flow of Judgels web applications.



The explanation of elements are:

- Global

  Global is the Global Object in Play Framework. It has some methods that can be overriden to handle requests.

  We use Global to construct DAOs, services, and controllers. By default, controllers in Play Framework are class with static methods. Controllers are designed to be stateless by that way. We change to construct it for dependency injection purposes.

- Controller

  Controller is the entry point of HTTP Request. HTTP Request trigger controller method call. We design controller to only check user permission and execute services. Controller can get business models from services as internal data for controller usage.

- Service

  Service is the layer that does all business logic of the system. Controller can call service to do some stuff and get business models. Service can call DAOs to do operation to databases or to get data models. By using services, we can create mock objects to do dependency injection for testing purposes.

- Data Access Object

  Data Access Object is an object that provides interface to do query to databases. The implementation depends on the database technologies that we used (in this case MySQL and Hibernate). By using DAOs, we can create mock objects to do dependency injection for testing purposes.

- Data Model

  Data Model represent the data structure used in the database. Every object in data model represent a row and field in data model represent a column in our cases. We use Java Persistence API annotations to annotate the constraints, table name, and other properties of the data model.

- Business Model

  Business Model represent the data structure that can be used by the controller. The data structure in business model doesn't have to be the same with data model. The main purpose of business model is to display data for users.

- View

  View is used to display information and to provide user way to interact with the system. We use Play Framework's default template engine which is Scala Template.

- Application Start

  Application Start is an event when the application started for the first time. In Play Framework, this event can be triggered by starting the application.

- HTTP Request

  HTTP Request is triggered when user open a web page either from browser or any other method.

- HTTP Resposne

  HTTP Response is the web page or any other content that are returned by the web application after processing the HTTP Request.

CHAPTER 12

Future Targets

We have some other things that we are planning for Judgels but is still in our minds and this document:

- Training Gate System

  A interactive training gate where anyone can learn about programming. The current training gate at tokilearning.org is still not friendly anough for anyone to start learning programming.

- Online Course System

  A system for academic purpose at academic institution (school, institute, university, etc). Through this system, institution can manage students by online means (like moodle). The key difference is this system support programming grading (auto grading).

- Control Panel

  Currently, Judgels system is still lack of user friendly way to control, manage and monitor running Judgels applications. Yeah, we need this.

- Forum

  We want to create forum which is integrated to Judgels where user can discuss and share anything related to Judgels. The forum can't use existing one because we need it to be integrated to Judgels.

- Drive

  Imagine when all of your source codes and your submissions is saved and tracked in online file manager. Well, that is the Drive for you.

- Editor

  The editor that can be used and integrated with Judgels system. The editor is like IDEOne, etc.