# autojudge

Jul 24, 2019

# Contents

CHAPTER 1

# Setting up and running `autojudge`

## 1.1 The autojudge "Install and Use" Reference

This section of the documentation will describe how to install the tool in your system / server and subsequently play around with it - basically how to use `autojudge`.

### 1.1.1 Installing `autojudge`

`autojudge` is a tool developed for automating evaluation for code submission in coding contests and in assigments at college. For your convenience, we have split this usage manual into 3 phases.

#### Phase 1 : Get `autojudge` and set up your environment

#### Phase 1a: Getting `autojudge`

`autojudge` is available on GitHub, and you can download a version of your choice from the releases page. We prefer that you use the latest version.

Extract the compressed files and you now have `autojudge` ready to use.

If you are a fan of `master`, then clone the repository, either using `git` or by downloading from GitHub from here.

#### Phase 1b: Setting up your environment

The evaluation of submissions are conducted on a Docker image that is built while initializing the application. Please install Docker using the instructions provided on their installation page.

If you are very conservative about populating your default environment with random Python packages, create a virtual environment for installing some *new* packages either using `virtualenv` or `conda-env`.

Install the requirements specified in `requirements.txt`. Don't forget to activate your environment if you have one.

If you going to deploy `autojudge`, please install PostgreSQL using the instructions provided on their installation page.

### Phase 2 : Run `autojudge`

Activate your environment.

Create and apply database migrations in Django with the following commands:

```
python manage.py makemigrations
python manage.py migrate
```

There are two ways of using `autojudge`.

#### Development

To run `autojudge` locally:

```
python manage.py runserver
```

Go to `localhost:8000` in your favourite browser. Keep yourself connected to internet for full functionality as certain frontend support such as JavaScript scripts are pulled from the internet.

The program `submission_watcher_saver.py` scores the submissions serially in the chronological order of submissions. It can be started anytime after the server has started, but it is preferred that the program be kept running in parallel with the server. Run it using:

```
python submission_watcher_saver.py
```

#### Production

The procedure to deploy `autojudge` is different from running locally. Below are a series of steps that will help you deploy `autojudge`.

Set the environmental variable `AUTOJUDGE_SECRET_KEY` to a random string, which should not be exposed to anyone. Think of it to be a private key.

Now modify a few more settings to `settings_production.py`. The first is to setup the database. We suggest using a PostgreSQL database. This modification can be done by adding the below dictionary to `settings_production.py`. Modify the values `NAME`, `USER`, `PASSWORD`, `HOST` and `PORT` accordingly.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',  # Sample
        'USER': 'mydatabaseuser',  # Sample
        'PASSWORD': 'mypassword',  # Sample
        'HOST': '127.0.0.1',  # Sample
        'PORT': '5432',  # Sample
    }
}
```

Next we setup the `STATIC_ROOT` path, the location where you would like the static files to be generated. This has to be set in `settings_production.py`.

To generate the static files, run:

```
python manage.py collectstatic --settings=autojudge.settings_production.py
```

The static files are generated in the path specified by `STATIC_ROOT` previously.

Now host the static files on a server and configure the URL in `STATIC_URL` in `settings_production.py`. If you have hosted the generated static files at https://static.autojudge.com, then change the `STATIC_URL` to https://static.autojudge.com/ (note the trailing slash is required).

You could optionally setup a cache server. Instructions to do this are specified here.

Configure the security settings in `settings_production.py` (leave it to the default values if you will be hosting on `https`).

To configure the Apache server using `WSGI`, follow the instructions here.

And finally, set environment variable `DJANGO_SETTINGS_MODULE` to `autojudge.settings_production` as opposed to `autojudge.settings` which is present by default.

### 1.1.2 User Manual for `autojudge`

#### Some important abstractions / terminology used in `autojudge`

---

**Note:** Please make note of the terms in **bold**

---

The judge works on graph between **contests** and **users**. A **contest** consists of a set of **problems**. A **user** is, well, a **user** - with different roles.

A user can be either a **poster**, **participant** or neither. A **user** is associated with the **contest** with one and only one role - either a **poster**, **participant** or neither.

The user who creates a new **contest** becomes the **poster** for the **contest** by default. This user can add more **posters** to help coordinate the **contest** (perhaps by setting new **problems**, verifying and commenting on **submissions**, and so on).

While creating a new **contest**, the first **poster** has an option to either allow select **participants**, or to leave it open for all. The former kind of a **contest** is a **private contest**, and the latter kind of a **contest** is a **public contest** (for obvious reasons). No **poster** is allowed to take part in a **contest** as a **participant** i.e., he/she cannot submit solutions.

If the **contest** is **public**, every user is either a **poster** or a **participant**. If the **contest** is **private**, a user can either be a **poster**, a **participant** or neither - in which case, he/she will not be permitted to participant in the **contest**.

Maybe a short example will help you understand if something is confusing. . . .

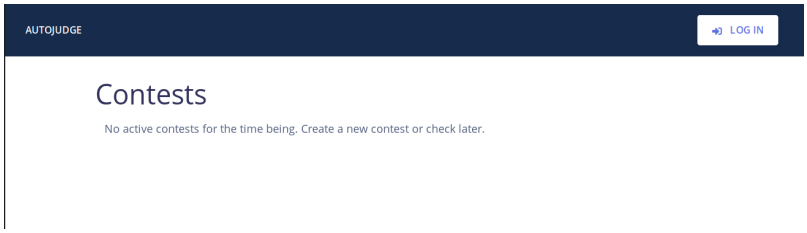
#### Example:

Take the case of a course assignment with programming questions. These programming questions could compose a **contest**, where each question is a **problem**. The instructor and the TAs can be thought of as the **posters**, while registered students for the course would be **participants**. Students not registered for the course will not be able to participate in this **contest** - as you would expect.

## Hands-on with `autojudge`

### Creating your account / Logging in

You need to be logged in to use `autojudge`. On the home page, click `LOG IN` (see the top right corner in the image below)

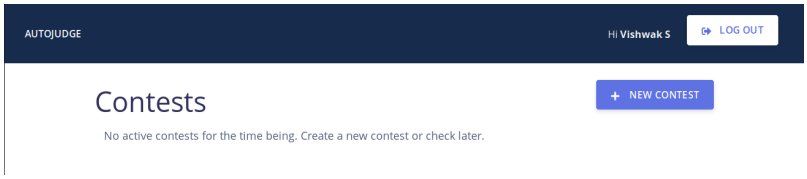



If this is being used at an institution, please make sure you log in with your institutional account. Currently, we support Google OAuth logins.

### Creating a contest

Once you are logged in, follow the steps below to create a new contest.

1. Click the `New Contest` button on the dashboard (see beneath the blue header in the image below)




2. Fill out the form for creating a new contest.

---

**Note:** The contest name distinguishes contests, hence every contest must have a unique name. Failure to provide a unique name will throw an interactive error.

---

**Note:** *Penalty* is a value between 0 and 1 and specifies the per day penalty on submissions made after *soft end date*. For example: a contest having 0.1 penalty for example, would give 90% of the actually scored points by a submission if it is made within 24 hours after *soft end date* but before *hard end date*.

---

**Note:** It is advised that *linter scoring* be disabled unless all code submissions are made in Python.

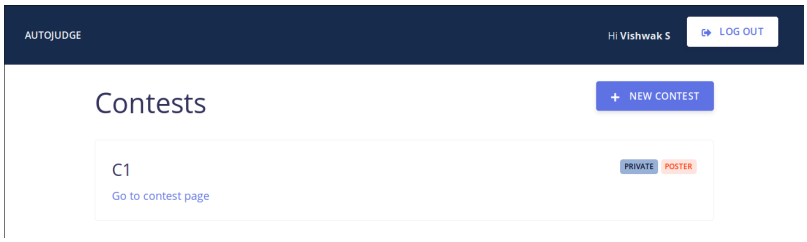
---

**Note:** Enable *poster scoring* if you would like the posters to give points in addition to those given by the judge.

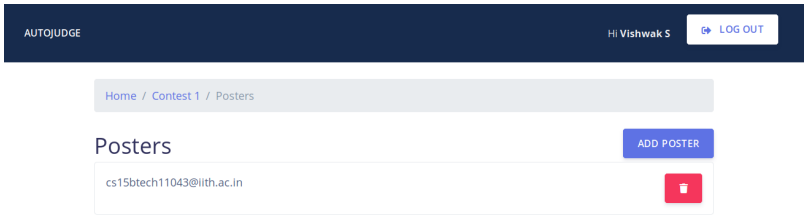
---

You should be able to see the newly created contest on your dashboard. No one else would be able to see this new contest on their dashboard until the start time of this contest.

Click on the contest in the link on the dashboard to edit it.




To add more posters to the contest, click on `SEE POSTERS`. You can add one or more posters by adding their emails in a comma separated list after clicking on `ADD POSTER`. The new poster(s) would now be able to see this contest on their dashboard (even before the start time). They can also edit the contest. To delete posters, click on the red bin button adjacent to each poster's email ID.




In the case of a private contest, the poster(s) can also see a `SEE PARTICIPANTS` button. Clicking this will lead them to a page where they can edit the `Participant` list in the same manner as the poster list.

---

**Note:** Trying to add a user both as a participant and a poster will not be permitted.

---

Any of the posters can update the dates of the contest by clicking on `UPDATE DATES`. Please update the dates before they pass, and attempting to do so will throw an interactive error.

Note that a participant cannot add or delete other participants or posters. Also he/she cannot update the dates.

A poster can also delete a contest using the button at the bottom of the contest page.

### Managing problems in a contest

A contest consists of problems. Problems can be added, edited or deleted by the posters of the contest.

A problem can be added to a contest only before the start time of the contest. To add a problem to the contest, follow the steps below:

1. Click `ADD PROBLEM` from the contest's homepage.
2. Fill the form that follows. Short descriptions for fields in the form are provided.

---

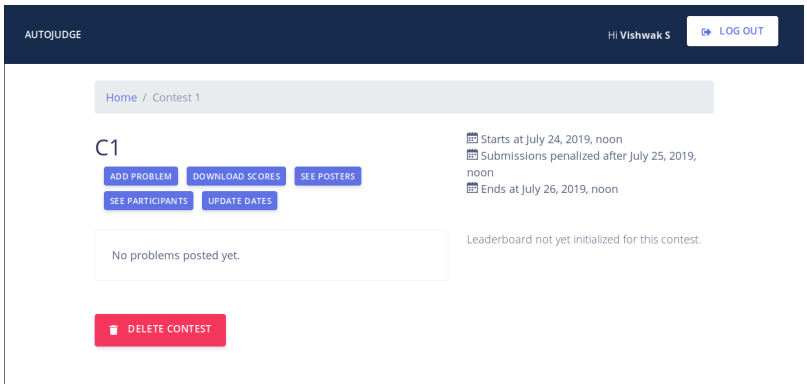
**Note:** The problem code distinctly identifies a problem, hence every problem must have a unique name. Failure to provide a unique name will throw an interactive error.

---

---

**Note:** In case the compilation script and test script are left empty, the default ones are used. The default scripts can be downloaded from the links just below the Browse button for each of them.

---




3. After submission, you can add or delete **test cases** on the problem page. There are two kinds of test cases - **public test cases** and **private test cases**. **Public test cases** would be visible to the participants while **private test cases** won't be visible.

---

**Note:** Test case addition and deletion will be allowed only till the start of the contest.

---

Posters can edit or delete an existing problem in the contest using the 2 icons on the top-right of the problem page (see to the right of the problem title).

---

**Note:** Deletion of a problem is only allowed until the contest begins.

---

### Submitting and checking submissions: Participant end

A participant can make **submission** for a problem from the problem page. Select the language and upload the submission file.

To check your previous **submissions**, and the judge's score for your **submissions**, click `SEE MY PREVIOUS SUBMISSIONS` at the bottom of the problem page.

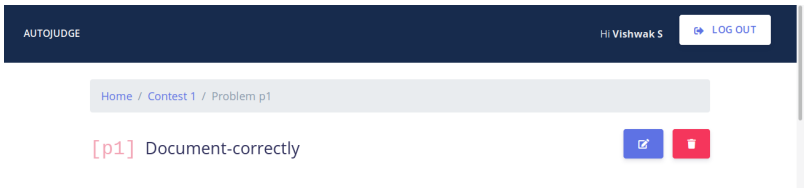
If you want a detailed verdict of the judge for a **submission**, click on that **submission**. You can see the verdict of the judge on individual **test cases** concisely below or in detail by clicking on a **test case**. You can also download your **submission** from here as well.

Once the contest begins and participants start submitting, the **leaderboard** is initialized and can be seen on the contest page. The leaderboard lists the participants in the decreasing order of sum of scores in individual problems in the contest.

Please note that the *max score* seen on the problem page is the maximum score possible per test case. For example, if there are 5 test cases and *max score* is 10 points, then a participant can score at most 50 points for that problem by the judge (i.e., notwithstanding the linter score and/or the poster score).

### Managing submissions from the poster's side

Posters can see all the **submissions** pertaining to a problem in the problem page by clicking `SEE ALL SUBMISSIONS` at the bottom of the page.




**Submissions** made by all the participants for a given problem would be available here. Click on any **submission** to open the **submission** page. The layout is the same as that seen by the participants.

In case *poster scoring* is enabled for the contest, the poster can give a score from the **submission** page by clicking on `UPDATE POSTER SCORE` on the top right adjacent to the download icon. Poster score can be any integer.

The poster can also view the submission file from the **submission** page by downloading it via the `DOWNLOAD` button on the top right.

### Commenting

Posters and participants can also comment. A **comment** by a participant to a problem can be viewed by all posters but not by any other participants - similar to private comments on Google Classroom.

To see old **comments** or create a new one, click on `SEE ALL SUBMISSIONS` on the problem page.

### Miscellaneous

A poster can download a CSV file containing the best scores of all participants in a contest by clicking on `DOWNLOAD SCORES` from the contest page.

CHAPTER 2

# The internals of the `judge` app in `autojudge`

## 2.1 The autojudge API Reference

This part of the documentation specifies routines and their description used in this project.

### 2.1.1 Models and Database Schema

#### Base Models

#### Contest

**class** judge.models.**Contest**(**args*, ***kwargs*)

Model for Contest.

**name**
Contest name

**start_datetime**
Start Date and Time for Contest

**soft_end_datetime**
"Soft" End Date and Time for Contest

**hard_end_datetime**
"Hard" End Date and Time for Contest

**penalty**
Penalty for late-submission

**public**
Is the contest public?

**enable_linter_score**
Enable linter scoring

**enable_poster_score**
Enable poster scoring

### Problem

**class** judge.models.**Problem**(**args*, ***kwargs*)
Model for a Problem.

**code**
Problem code

**contest**
Foreign key to contest for the problem

**name**
Problem name

**statement**
Problem statement

**input_format**
Problem input format

**output_format**
Problem output format

**difficulty**
Problem difficulty

**time_limit**
Problem time limit

**memory_limit**
Problem memory limit

**file_exts**
Accepted file extensions for submissions to problem

**starting_code**
Problem starting code

**max_score**
Maximum score for a test case for the problem

**compilation_script**
Problem compilation script

**test_script**
Problem test script

### Submission

**class** judge.models.**Submission**(**args*, ***kwargs*)
Model for a Submission.

**problem**
Foreign key to problem for which this is a submission

**participant**
Foreign key to person who submitted the solution

**file_type**
File type of submission

**submission_file**
Submission file

**timestamp**
Timestamp of submission

**judge_score**
Judge score

**poster_score**
Poster score

**linter_score**
Linter score

**final_score**
Final score

### TestCase

**class** judge.models.**TestCase**(**args*, ***kwargs*)
Model for TestCase. Maintains testcases and mapping between TestCase and Problem.

**problem**
Foreign key to problem for which this is a test case

**public**
Determines if the test case is a public test case or a private test case

**inputfile**
Input file for the test case

**outputfile**
Output file for the test case

### Person

**class** judge.models.**Person**(**args*, ***kwargs*)
Model for Person.

**email**
Email ID of the Person

**rank**
Rank of the Person

### Comment

**class** judge.models.**Comment**(**args*, ***kwargs*)
Model for Comment.

**problem**
Foreign key to problem relating to the comment

**person**
Foreign key to person

**commenter**
Foreign key to person who commented

**timestamp**
Timestamp of the comment

**comment**
Content of the comment

### Derived Models

### ContestPerson

**class** judge.models.**ContestPerson**(**args*, ***kwargs*)
Model for ContestPerson. This maps how (either as a Participant or Poster) persons have access to the contests.

**contest**
Foreign key to contest in which this person is taking part

**person**
Foreign key to the actual person

**role**
Determines if Person is a Poster or a Participant

### SubmissionTestCase

**class** judge.models.**SubmissionTestCase**(**args*, ***kwargs*)
Model for SubmissionTestCase. Maintains mapping between TestCase and Submission.

**submission**
Foreign key to submission

**testcase**
Foreign key to test case

**verdict**
Verdict by the judge

**memory_taken**
Virtual memory consumed by the submission

**time_taken**
Time taken by the submission

**message**
Message placeholder, used for erroneous submissions

### PersonProblemFinalScore

**class** judge.models.**PersonProblemFinalScore**(**args*, ***kwargs*)
Model to store the final score assigned to a person for a problem.

**problem**
Foreign key to problem for which the score is saved

**person**
Foreign key to person whose submission's score is saved

**score**
Final score saved

### 2.1.2 Forms and input pre-processing

#### Creation forms

#### NewContestForm

**class** judge.forms.**NewContestForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form for creating a new Contest.

**contest_name = None**
Contest Name

**contest_start = None**
Contest Start Timestamp

**contest_soft_end = None**
Contest Soft End Timestamp

**contest_hard_end = None**
Contest Hard End Timestamp

**penalty = None**
Contest Penalty factor

**is_public = None**
Contest is_public property

**enable_linter_score = None**
Contest enable_linter_score property

**enable_poster_score = None**
Contest enable_poster_score property

**clean**()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

### NewProblemForm

**class** judge.forms.**NewProblemForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form for adding a new Problem.

**code = None**
Problem Code Field

**name = None**
Problem Name Field

**statement = None**
Problem Statement Field

**input_format = None**
Problem Input Format Field

**output_format = None**
Problem Output Format Field

**difficulty = None**
Problem Difficulty Field

**time_limit = None**
Problem Time limit

**memory_limit = None**
Problem Memory limit

**file_exts = None**
Problem File Extensions

**starting_code = None**
Problem Starting code

**max_score = None**
Problem Max Score

**compilation_script = None**
Problem Compilation Script

**test_script = None**
Problem Test Script

### NewSubmissionForm

**class** judge.forms.**NewSubmissionForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form to create a new Submission.

**file_type = None**
Choices of file type

**submission_file = None**
Submission File

### NewCommentForm

**class** judge.forms.**NewCommentForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form to add a new comment

**participant_email = None**
Email of participant

**comment = None**
Comment content

### Extension forms

### AddPersonToContestForm

**class** judge.forms.**AddPersonToContestForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form to add a Person to a Contest.

**emails = None**
Email ID of the person

### AddTestCaseForm

**class** judge.forms.**AddTestCaseForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form to create a new TestCase

**test_type = None**
TestCase Type

**input_file = None**
TestCase Input

**output_file = None**
TestCase Output

### AddPosterScoreForm

**class** judge.forms.**AddPosterScoreForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form to add poster score for a submission

**score = None**
Score field

### Updation forms

### UpdateContestForm

**class** judge.forms.**UpdateContestForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form to update the timeline of the Contest

**contest_start = None**
Contest Start Timestamp

**contest_soft_end = None**
Contest Soft End Timestamp

**contest_hard_end = None**
Contest Hard End Timestamp

**clean**()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

#### EditProblemForm

**class** judge.forms.**EditProblemForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form for editing an existing problem.

**name = None**
Problem Name Field

**statement = None**
Problem Statement Field

**input_format = None**
Problem Input Format Field

**output_format = None**
Problem Output Format Field

**difficulty = None**
Problem Difficulty Field

#### Deletion forms

#### DeletePersonFromContestForm

**class** judge.forms.**DeletePersonFromContestForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *field_order=None*, *use_required_attribute=None*, *renderer=None*)

Form to remove a Person from a Contest.

**email = None**
Email ID of the person

### 2.1.3 Views and page rendering

#### Default Views

judge.views.**index**(*request*)
Renders the index page.

**Parameters** **request** (*HttpRequest*) – the request object used

judge.views.**handler404**(*request*, **args*)
Renders 404 page.

**Parameters** **request** (*HttpRequest*) – the request object used

judge.views.**handler500**(*request*, **args*)
Renders 500 page.

**Parameters** **request** (*HttpRequest*) – the request object used

### Creation Views

judge.views.**new_contest**(*request*)
Renders view for the page to create a new contest.

**Parameters** **request** (*HttpRequest*) – the request object used

judge.views.**new_problem**(*request*, *contest_id*)
Renders view for the page to create a new problem in a contest.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

### Modification Views

judge.views.**edit_problem**(*request*, *problem_id*)
Renders view for the page to edit selected fields of a pre-existing problem.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID

judge.views.**add_person**(*request*, *contest_id*, *role*)
Function to render the page for adding a person - participant or poster to a contest.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID
- **role** (*bool*) – `True` for Poster, `False` for Participant

judge.views.**add_poster**(*request*, *contest_id*)
Renders the page for adding a poster. Dispatches to *add_person()* with `role` set to `True`.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

judge.views.**add_participant**(*request*, *contest_id*)
Renders the page for adding a participant. Dispatches to *add_person()* with `role` set to `False`.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

### Detail Views

judge.views.**contest_detail**(*request*, *contest_id*)
Renders the contest preview page after the contest has been created.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

judge.views.**problem_detail**(*request*, *problem_id*)
Renders the problem preview page after the problem has been created. This preview will be changed based on the role of the user (poster or participant).

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID

judge.views.**submission_detail**(*request*, *submission_id*)
Renders the page where a detailed breakdown with respect to judge's evaluation, additional scores, error messages displayed and so on.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **submission_id** (*str*) – the submission ID

judge.views.**get_people**(*request*, *contest_id*, *role*)
Function to render the page for viewing participants and posters for a contest based on `role`.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID
- **role** (*bool*) – `True` for Poster, `False` for Participant

judge.views.**get_posters**(*request*, *contest_id*)
Renders the page for posters of a contest. Dispatches to *get_people()* with `role` set to `True`.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

judge.views.**get_participants**(*request*, *contest_id*)
Renders the page for posters of a contest. Dispatches to *get_people()* with `role` set to `False`.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

judge.views.**problem_submissions**(*request*, *problem_id*)
Renders the page where all submissions to a given problem can be seen. For posters, this renders a set of tables for each participant. For participants, this renders a table with the scores of their submissions only.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID

### Deletion Views

judge.views.**delete_contest**(*request*, *contest_id*)
Function to provide the option to delete a contest.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

judge.views.**delete_problem**(*request*, *problem_id*)
Function to provide the option to delete a problem.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID

judge.views.**delete_testcase**(*request*, *problem_id*, *testcase_id*)
Function to provide the option to delete a test-case of a particular problem.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID
- **testcase_id** (*str*) – the testcase ID

### Downloading Views

judge.views.**contest_scores_csv**(*request*, *contest_id*)
Function to provide the facility to download a CSV of scores of participants in a contest at a given point in time.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **contest_id** (*int*) – the contest ID

judge.views.**problem_starting_code**(*request*, *problem_id*)
Function to provide the facility to download the starting code for a problem.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID

judge.views.**problem_compilation_script**(*request*, *problem_id*)
Function to provide the facility to download the compilation script for a problem after creating the problem.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID

judge.views.**problem_test_script**(*request*, *problem_id*)
Function to provide the facility to download the testing script for a problem after creating the problem.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **problem_id** (*str*) – the problem ID

judge.views.**problem_default_script**(*request*, *script_name*)
Function to provide the facility to download the default compilation or test script.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **script_name** (*str*) – name of the script - one of *compilation_script* or *test_script*

judge.views.**submission_download**(*request*, *submission_id*)
Function to provide the facility to download a given submission.

**Parameters**

- **request** (*HttpRequest*) – the request object used
- **submission_id** (*str*) – the submission ID

### 2.1.4 Handlers and database management

#### Process Functions

judge.handler.**process_contest**(*contest_name*, *contest_start*, *contest_soft_end*, *contest_hard_end*, *penalty*, *is_public*, *enable_linter_score*, *enable_poster_score*)
Function to process a new *Contest*.

**Parameters**

- **contest_name** (str) – Name of the contest
- **contest_start** (datetime) – A *datetime* object representing the beginning of the contest
- **contest_soft_end** (datetime) – A *datetime* object representing the soft deadline of the contest
- **contest_hard_end** (datetime) – A *datetime* object representing the hard deadline of the contest
- **penalty** (float) – A penalty score for late submissions
- **is_public** (bool) – Field to indicate if the contest is public (or private)
- **enable_linter_score** (bool) – Field to indicate if linter scoring is enabled in the contest
- **enable_poster_score** (bool) – Field to indicate if poster scoring is enabled in the contest

**Return type** Tuple[bool, Union[ValidationError, str]]

**Returns** A 2-tuple - 1st element indicating whether the processing has succeeded, and 2nd element providing a `ValidationError` if processing is unsuccessful.

`judge.handler.`**`process_problem`**(*contest_id*, ***kwargs*)

Function to process a new *Problem*.

**Parameters** **`contest_id`** (`int`) – Contest ID to which the problem belongs

`**kwargs` includes the following keyword arguments, which are directly passed to the construct a *Problem* object.

**Parameters**

- **`code`** (*str*) – Problem code
- **`name`** (*str*) – Problem name
- **`statement`** (*Optional[InMemoryUploadedFile]*) – Problem statement
- **`input_format`** – Problem input format
- **`output_format`** – Problem output format
- **`difficulty`** – Problem difficulty
- **`time_limit`** – Problem execution time limit
- **`memory_limit`** – Problem virtual memory limit
- **`file_exts`** – Accepted file format for submissions
- **`starting_code`** – Starting code for the problem
- **`max_score`** – Maximum judge score per test case for the problem
- **`compilation_script`** – Compilation script for the submissions
- **`test_script`** – Test script for the submissions

**Return type** `Tuple[bool, Optional[ValidationError]]`

**Returns** A 2-tuple - 1st element indicating whether the processing has succeeded, and 2nd element providing a `ValidationError` if processing is unsuccessful.

`judge.handler.`**`process_submission`**(*problem_id*, *participant_id*, *file_type*, *submission_file*, *timestamp*)

Function to process a new *Submission* for a problem by a participant.

**Parameters**

- **`problem_id`** (`str`) – Problem ID for the problem corresponding to the submission
- **`participant_id`** (`str`) – Participant ID
- **`file_type`** (`str`) – Submission file type
- **`submission_file`** (`InMemoryUploadedFile`) – Submission file
- **`timestamp`** (`str`) – Time at submission

**Return type** `Tuple[bool, Optional[ValidationError]]`

**Returns** A 2-tuple - 1st element indicating whether the processing has succeeded, and 2nd element providing a `ValidationError` if processing is unsuccessful.

judge.handler.**process_testcase**(*problem_id*, *test_type*, *input_file*, *output_file*)
Function to process a new *TestCase* for a problem.

> **Warning:** This function does not rescore all the submissions and so score will not change in response to the new testcase. Do not call this function once the contest has started, it will lead to erroneous scores.

**Parameters**

- **problem_id** (str) – Problem ID to which the testcase is added.
- **test_type** (str) – Type of testcase - one of *public*, *private*.
- **input_file** (InMemoryUploadedFile) – Input file for the testcase.
- **output_file** (InMemoryUploadedFile) – Output file for the testcase.

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the processing has succeeded, and 2nd element providing a ValidationError if processing is unsuccessful.

judge.handler.**process_person**(*email*, *rank=0*)
Function to process a new *Person*.

**Parameters**

- **email** (str) – Email of the person
- **rank** (int) – Rank of the person (defaults to 0).

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the processing has succeeded, and 2nd element providing a ValidationError if processing is unsuccessful.

judge.handler.**process_comment**(*problem_id*, *person_id*, *commenter_id*, *timestamp*, *comment*)
Function to process a new *Comment* on the problem.

**Parameters**

- **problem_id** (str) – Problem ID
- **person_id** (str) – Person ID
- **commenter_id** (str) – Commenter (another person) ID
- **timestamp** (datetime) – Date and Time of comment
- **comment** (str) – Comment content

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the processing has succeeded, and 2nd element providing a ValidationError if processing is unsuccessful.

### Addition Functions

judge.handler.**add_person_to_contest**(*person_id*, *contest_id*, *permission*)
Function to relate a person to a contest with permissions.

**Parameters**

- **person_id** (str) – Person ID
- **contest_id** (int) – Contest ID
- **permission** (bool) – If True, then poster, if False, then participant

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the addition has succeeded, and 2nd element providing a ValidationError if addition is unsuccessful.

judge.handler.**add_persons_to_contest**(*persons*, *contest_id*, *permission*)

Function to relate a list of persons and contest with permissions. This function would create records for all the persons who are not present in the database irrespective of whether anyone has conflict or not.

**Parameters**

- **persons** (List[str]) – List of person IDs
- **contest_id** (int) – Contest ID
- **permission** (bool) – If True, then poster, if False, then participant

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the relation creation has succeeded, and 2nd element providing a ValidationError if relation creation is unsuccessful.

### Update Functions

judge.handler.**update_problem**(*code*, *name*, *statement*, *input_format*, *output_format*, *difficulty*)

Function to update selected fields in a *Problem* after creation. The fields that can be modified are *name*, *statement*, *input_format*, *output_format* and *difficulty*.

**Parameters**

- **code** (str) – Problem ID
- **name** (str) – Modified problem name
- **statement** (str) – Modified problem statement
- **input_format** (str) – Modified problem input format
- **output_format** (str) – Modified problem output format
- **difficulty** (str) – Modified problem difficulty

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the update has succeeded, and 2nd element providing a ValidationError if update is unsuccessful.

judge.handler.**update_poster_score**(*submission_id*, *new_score*)

Function to update the poster score for a submission. Leaderboard is updated if the total score for the person-problem pair has changed.

**Parameters**

- **submission_id** (str) – Submission ID of the submission
- **new_score** (int) – New score to be assigned

**Returns** A 2-tuple - 1st element indicating whether the update has succeeded, and 2nd element providing a `ValidationError` if update is unsuccessful.

`judge.handler.`**`update_leaderboard`**(*contest_id*, *person_id*)
Function to update the leaderboard for a person-contest pair given their IDs.

---

**Note:** Only call this function when some submission for some problem of the contest has scored more than its previous submission. Remember to call this function whenever *`PersonProblemFinalScore`* is updated.

---

**Parameters**

- **`contest_id`** (`int`) – Contest ID
- **`person_id`** (`str`) – Person ID

**Return type** `bool`

**Returns** If update is successful, then `True`. If unsuccessful, then `False`.

### Getter Functions

`judge.handler.`**`get_personcontest_permission`**(*person_id*, *contest_id*)
Function to give the relation between a *`Person`* and a *`Contest`*.

**Parameters**

- **`person_id`** (`Optional[str]`) – Person ID
- **`contest_id`** (`int`) – Contest ID

**Return type** `Optional[bool]`

**Returns** If participant, then `False`, if poster, then `True`, if neither, then `None`

`judge.handler.`**`get_personproblem_permission`**(*person_id*, *problem_id*)
Function to give the relation between a *`Person`* and a *`Contest`*. This dispatches to *`get_personcontest_permission()`* with relevant arguments.

**Parameters**

- **`person_id`** (`Optional[str]`) – Person ID
- **`problem_id`** (`str`) – Problem ID

**Return type** `Optional[bool]`

**Returns** If participant, then `False`, if poster, then `True`, if neither, then `None`

`judge.handler.`**`get_posters`**(*contest_id*)
Function to return the list of the posters for a *`Contest`*.

**Parameters** **`contest_id`** (`int`) – Contest ID

**Return type** `Tuple[bool, Union[ValidationError, List[str]]]`

**Returns** A 2-tuple - 1st element indicating whether the retrieval has succeeded. If successful, a list of IDs are present in the 2nd element. If unsuccessful, a `ValidationError` is additionally returned.

`judge.handler.`**`get_participants`**(*contest_id*)
Function to return the list of the participants for a *`Contest`*.

**Parameters** **contest_id** (int) – Contest ID

**Return type** Tuple[bool, Union[ValidationError, List[str]]]

**Returns** A 2-tuple - 1st element indicating whether the retrieval has succeeded. If successful, a list of IDs are present in the 2nd element. The list is empty if the contest is public. If unsuccessful, a ValidationError is additionally returned.

judge.handler.**get_personcontest_score**(*person_id*, *contest_id*)

Function to get the final score, which is the sum of individual final scores of all problems in a contest for a particular person.

**Parameters**

- **person_id** (str) – Person ID
- **contest_id** (int) – Contest ID

**Return type** Tuple[bool, Union[float, ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the retrieval has succeeded. If successful, the final score is present in the 2nd element. If unsuccesful, a ValidationError is additionally returned.

judge.handler.**get_submission_status**(*submission_id*)

Function to get the current status of the submission given its submission ID.

**Parameters** **submission_d** – Submission ID

**Returns** A 2-tuple - 1st element indicating whether the retrieval has succeeded. If successful, a tuple consisting of a dictionary and a smaller tuple. The key for the dictionary is the testcase ID, and value is another smaller tuple consisting of the verdict, time taken, memory consumed, flag to indicate if the testcase was public or private and message after checking. The smaller tuple consists of the score given by the judge, poster (if applicable), and linter (if applicable), as well as the final score, timestamp of submission and the file type of submission. If unsuccessful, a ValidationError is additionally returned.

judge.handler.**get_submissions**(*problem_id*, *person_id*)

Function to retrieve all submissions made by everyone or a specific person for this problem.

**Parameters**

- **problem_id** (str) – Problem ID
- **person_id** (Optional[str]) – Person ID

**Return type** Tuple[bool, Union[Dict[str, List[Any]], ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the retrieval has succeeded. If successful, and person_id is None, then the list of submissions pertaining to each person is placed in a dictionary, and if person_id is provided, then the list of submissions pertaining to the specific person is placed in a dictionary and returned. If unsuccessful, then a ValidationError is additionally returned.

judge.handler.**get_leaderboard**(*contest_id*)

Function to returns the current leaderboard for a contest given its contest ID.

**Parameters** **contest_id** (int) – Contest ID

**Return type** Tuple[bool, Union[str, List[List[Union[str, float]]]]]

**Returns** A 2-tuple - 1st element indicating whether leaderboard has been initialized or not. If initialized, a list of 2-length lists is returned ordered by decreasing scores. The

first element is the rank, and the second element is the score. If uninitialized, a suitable message is provided

judge.handler.**get_comments**(*problem_id*, *person_id*)
Function to get the private comments on the problem for the person.

**Parameters**

- **problem_id** (str) – Problem ID
- **person_id** (str) – Person ID

**Return type** List[Tuple[Any, Any, Any]]

**Returns** List of 3-tuple of comments - the person who commented, the timestamp and the comment content, sorted in chronological order.

judge.handler.**get_csv**(*contest_id*)
Function to get the CSV (in string form) of the current scores of all participants in a contest given its contest ID.

**Parameters** **contest_id** (int) – Contest ID

**Return type** Tuple[bool, Union[ValidationError, StringIO]]

**Returns** A 2-tuple - 1st element indicating whether the retrieval has succeeded, and 2nd element providing a ValidationError if processing is unsuccessful or a StringIO object if successful.

### Deletion Functions

judge.handler.**delete_contest**(*contest_id*)
Function to delete a *Contest* given its contest ID. This will cascade delete in all the tables that have contest_id as a foreign key. It calls *delete_problem()* for each problem in the contest.

**Parameters** **contest_id** (int) – the contest ID

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the deletion has succeeded, and 2nd element providing a ValidationError if deletion is unsuccessful.

judge.handler.**delete_problem**(*problem_id*)
Function to delete a *Problem* given its problem ID. This will cascade delete in all the tables that have problem_id as a foreign key. It will also delete all the submissions, testcases and related directories corresponding to the problem.

**Parameters** **problem_id** (str) – the problem ID

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the deletion has succeeded, and 2nd element providing a ValidationError if deletion is unsuccessful.

judge.handler.**delete_testcase**(*testcase_id*)
Function to delete a *TestCase* given its testcase ID. This will cascade delete in all the tables where this testcase appears.

> **Warning:** This function does not rescore all the submissions and so score will not change in response to the deleted testcase. Do not call this function once the contest has started, it will lead to erroneous scores.

**Parameters** **testcase_id** (str) – the testcase ID

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the deletion has succeeded, and 2nd element providing a ValidationError if deletion is unsuccessful.

judge.handler.**delete_personcontest**(*person_id*, *contest_id*)
Function to delete the relation between a person and a contest.

**Parameters**

- **person_id** (str) – Person ID
- **contest_id** (int) – Contest ID

**Return type** Tuple[bool, Optional[ValidationError]]

**Returns** A 2-tuple - 1st element indicating whether the deletion has succeeded, and 2nd element providing an error message if deletion is unsuccessful.

# Python Module Index

## j

# Index

## R



## S



## T



## U



## V