grand-challenge.org Documentation

James Meakin

Mar 03, 2020

Contents:

1 Getting Started **1**
- 1.1 Installation . . . 1
- 1.2 Running the Tests . . . 2
- 1.3 Development . . . 3
- 1.4 Creating Migrations . . . 5
- 1.5 Building the documentation . . . 5
- 1.6 Adding new dependencies . . . 5
- 1.7 Going to Production . . . 5

2 Evaluation **7**
- 2.1 Evaluation Container Requirements . . . 7
- 2.2 Template Tags . . . 9

3 Processor specification **11**
- 3.1 What is a processor? . . . 11
- 3.2 Processor Inputs . . . 11
- 3.3 Execution . . . 12
- 3.4 Processor Outputs . . . 12
- 3.5 Runtime requirements . . . 13

4 Workstations **15**
- 4.1 Models . . . 15

5 Reader Studies **19**
- 5.1 Creating a Reader Study . . . 19
- 5.2 Cases . . . 19
- 5.3 Defining the Hanging List . . . 20
- 5.4 Questions . . . 20
- 5.5 Adding Ground Truth . . . 20

6 Design decisions **25**
- 6.1 Definitions . . . 25
- 6.2 Image database objects . . . 25

7 Indices and tables **27**

Python Module Index **29**

Index 31

CHAPTER 1

Getting Started

Grand-challenge is distributed as a set of containers that are defined and linked together in `docker-compose.yml`. To develop the platform you need to have docker and docker-compose running on your system.

1.1 Installation

1. Download and install Docker

   Linux: Docker and Docker Compose

   Windows 10 Pro (Build 15063 or later): Docker for Windows

   Older Windows versions: Docker Toolbox

2. Clone the repo

```
$ git clone https://github.com/comic/grand-challenge.org
$ cd grand-challenge.org
```

3. You can then start the site by invoking

```
$ ./cycle_docker_compose.sh
```

You can then navigate to https://gc.localhost in your browser to see the development site, this is using a self-signed certificate so you will need to accept the security warning. The `app/` directory is mounted in the containers, `werkzeug` handles the file monitoring and will restart the process if any changes are detected. If you need to manually restart the process you can do this when running `cycle_docker_compose.sh` by pressing `CTRL+D` in the console window, you can also kill the server with `CTRL+C`.

1.1.1 Windows

Running Grand-Challenge within a Windows environment requires additional steps before invoking the `cycle_docker_compose.sh` script.

1. Install `Make` for an available `bash` console
2. Set an environment variable to enable Windows path conversions for Docker

```
$ export COMPOSE_CONVERT_WINDOWS_PATHS=1
```

3. Add the following line to your hosts file (`C:\Windows\System32\drivers\etc\hosts`)

```
# Using Docker for Windows:
127.0.0.1 gc.localhost

# Using Docker Toolbox:
192.168.99.100 gc.localhost
```

4. Share the drive where this repository is located with Docker

   Docker for Windows

   1. Right-click Docker icon in taskbar and click "Settings"
   2. Go to "Shared drives" tab
   3. Mark the checkbox of the drive where this repository is located

   Docker Toolbox

   1. Open VirtualBox
   2. Go to the virtual machine that belongs to docker
   3. Double click "Shared folders"
   4. Click on the "Add New Shared Folder" button on the right
   5. In the Folder Path box, type the drive letter where this repository is located (eg. `C:\`)
   6. In the Folder Name box, type the drive letter lowercased (eg. `c`)
   7. Restart the docker machine by typing `docker-machine restart` in your console
   8. SSH into the docker VM with `docker-machine ssh`
   9. Append the following lines to the file `/mnt/sda1/var/lib/boot2docker/profile`

```
mkdir /home/docker/c # Change the 'c' to your drive letter
sudo mount -t vboxsf -o uid=1000,gid=50 c /home/docker/c # Again, change both 'c's to␣
↪your drive letter
```

1.2 Running the Tests

GitHub actions is used to run the test suite on every new commit. You can also run the tests locally by

1. In a console window make sure the database is running

```
$ ./cycle_docker_compose.sh
```

2. Then in a second window run

```
$ docker-compose run --rm web pytest -n 2
```

Replace 2 with the number of CPUs that you have on your system, this runs the tests in parallel.

If you want to add a new test please add them to the `app/tests` folder. If you only want to run the tests for a particular app, eg. for `teams`, you can do

```
$ docker-compose run --rm web pytest -k teams_tests
```

1.3 Development

You will need to install pre-commit so that the code is correctly formatted

```
$ python3 -m pip install pre-commit
```

Please do all development on a branch and make a pull request to master, this will need to be reviewed before it is integrated.

We recommend using Pycharm for development.

1.3.1 Running through docker-compose

You will need the Professional edition to use the docker-compose integration. To set up the environment in Pycharm Professional 2018.1:

1. `File` -> `Settings` -> `Project: grand-challenge.org` -> `Project Interpreter` -> `Cog` wheel (top right) -> `Add` -> `Docker Compose`
2. Then select the docker server (usually the unix socket)
3. Set the service to `web`
4. Click `OK`
5. Set the path mappings from `<Project root>/app->/app`
6. Click `OK`

Pycharm will then spend some time indexing the packages within the container to help with code completion and inspections. If you edit any files these will be updated on the fly by werkzeug.

1.3.2 PyCharm Configuration

It is recommended to setup django integration to ensure that the code completion, tests and import optimisation works.

1. Open `File` -> `Settings` -> `Languages and Frameworks` -> `Django`
2. Check the `Enable Django Support` checkbox
3. Set the project root to `<Project root>/app`
4. Set the settings to `config/settings.py`
5. Check the `Do not use the django test runner` checkbox
6. In the settings window navigate to `Tools` -> `Python integrated tools`
7. Under the testing section select `pytest` as the default test runner
8. Under the Docstrings section set `NumPy` as the docstrings format
9. In the settings window navigate to `Editor` -> `Code Style`

10. Click on the `Formatter Control` tab and enable `Enable formatter markers in comments`
11. In the settings window navigate to `Editor` -> `Code Style` -> `Python`
12. On the `Wrapping and Braces` tab set `Hard wrap at` to `86` and `Visual guide` to `79`
13. On the `Imports` tab enable `Sort Import Statements`, `Sort imported names in "from" imports`, and `Sort plain and "from" imports separately in the same group`
14. Click `OK`
15. Install the `Flake8 Support` plugin so that PyCharm will understand `noqa` comments

It is also recommended to install the black extension (version `19.10b0`) for code formatting.

1.3.3 Running locally

Alternatively, it can be useful to run code from a local python environment - this allows for easier debugging and does not require e.g. the professional edition of PyCharm. The setup described here uses all services from the normal `docker-compose` stack, except for the web service. Though this service is running, a separate Django dev server is started in PyCharm (or from the terminal). As the dev server is running on port `8000` by default, there is no port conflict with the service running in the docker container.

1. Run the `docker-compose` stack for the database and celery task handling

```
$ ./cycle_docker_compose.sh
```

2. Make sure you have `poetry` installed.
3. In a new terminal, create a new virtual python environment using `poetry install` in this repository's root folder.
4. Activate the virtual env: `poetry shell`.
5. Load the environmental variables contained in `.env.local`

```
$ export $(cat .env.local | egrep -v "^#" | xargs)
```

6. Run migrations and check_permissions (optionally load demo data).

```
$ cd app
$ python manage.py migrate
$ python manage.py check_permissions
$ python manage.py init_gc_demo
```

7. You can now start the server using `python manage.py runserver_plus`.
8. To setup PyCharm:
    1. `File` -> `Settings` -> `Project: grand-challenge.org` -> `Project Interpreter` -> Select your created virtual environment
    2. For each run/debug configuration, make sure the environmental variables are loaded, the easiest is to use this plugin. Or they can be pasted after pressing the folder icon in the `Environmental variables` field.
    3. Useful to setup: the built-in python/django console in Pycharm: `Settings` -> `Build, execution, deployment` -> `Console` -> Python/Django console. Choose the same python interpreter here, and make sure to load the environmental variables (the .env plugin cannot be used here, the variables can only be pasted).

1.4 Creating Migrations

If you change a `models.py` file then you will need to make the corresponding migration files. You can do this with

```
$ make migrations
```

or, more explicitly

```
$ docker-compose run --rm --user `id -u` web python manage.py makemigrations
```

add these to git and commit.

1.5 Building the documentation

1.5.1 Using docker

Having built the web container with `cycle_docker_compose.sh` you can use this to generate the docs with

```
$ make docs
```

This will create the docs in the `docs/_build/html` directory.

1.6 Adding new dependencies

Poetry is used to manage the dependencies of the platform. To add a new dependency use

```
$ poetry add <whatever>
```

and then commit the `pyproject.toml` and `poetry.lock`. If this is a development dependency then use the `--dev` flag, see the `poetry` documentation for more details.

Versions are unpinned in the `pyproject.toml` file, to update the resolved dependencies use

```
$ poetry lock
```

and commit the update `poetry.lock`. The containers will need to be rebuilt after running these steps, so stop the `cycle_docker_compose.sh` process with `CTRL+C` and restart.

1.7 Going to Production

The docker compose file included here is for development only. If you want to run this in a production environment you will need to make several changes, not limited to:

1. Use `gunicorn` rather than run `runserver_plus` to run the web process
2. Disable mounting of the docker socket
3. Removing the users that are created by `init_gc_demo`

CHAPTER 2

Evaluation

grand-challenge.org has a system for automatically evaluating new submissions. Challenge administrators upload their own Docker containers that will be executed by Celery workers when a new submission in uploaded by a participant.

2.1 Evaluation Container Requirements

The evaluation container must contain everything that is needed to perform the evaluation on a new submission. This includes the reference standard, and the code that will execute the evaluation on the new submission. An instance of the evaluation container image is created for each submission.

2.1.1 Input

The participants submission will be extracted and mounted as a docker volume on */input/*.

2.1.2 Entrypoint

The container will be run with the default arguments, so the entrypoint must by default produce an evaluation for the data that will reside on */input/*. The container is responsible for loading all of the data, handling incorrect filenames, incomplete submissions, duplicate folders, etc.

2.1.3 Errors

If there is an error in the evaluation process grand-challenge.org will parse *stderr* and return the last non-empty line to the user. If your evaluation script is in Python the best practice is to raise an exception and the message will then be passed to the user, eg

```
raise AttributeError('Expected to find 10 images, you submitted 5')
```

2.1.4 Output

The container must produce the file */output/metrics.json*. The structure within must be valid json (ie. loadable with *json.loads()*) and will be stored as a result in the database. The challenge administrator is free to define what metrics are included. We recommend storing results in two objects - *case* for the scores on individual cases (eg, scans), and *aggregates* for when there is one number per evaluation. For example:


```
{
  "case": {
    "dicecoefficient": {
      "0": 0.6461774875144065,
      "1": 0.7250400040547097,
      "2": 0.6747092236948878,
      "3": 0.6452332692745784,
      "4": 0.6839602948067993,
      "5": 0.6817807628480707,
      "6": 0.4715406247268339,
      "7": 0.5988810496224731,
      "8": 0.5475856316815167,
      "9": 0.32923801642370615
    },
    "jaccardcoefficient": {
      "0": 0.47729852440408627,
      "1": 0.5686766693547471,
      "2": 0.5091027839007266,
      "3": 0.47626890640360103,
      "4": 0.5197109875240358,
      "5": 0.5171983108978807,
      "6": 0.30850713624139353,
      "7": 0.4274305543159676,
      "8": 0.3770174983296798,
      "9": 0.1970585994056237
    },
    "alg_fname": {
      "0": "1.2.840.113704.1.111.2296.1199810886.7.mhd",
      "1": "1.2.276.0.28.3.0.14.4.0.20090213134050413.mhd",
      "2": "1.2.276.0.28.3.0.14.4.0.20090213134114792.mhd",
      "3": "1.2.840.113704.1.111.2004.1131987870.11.mhd",
      "4": "1.2.840.113704.1.111.2296.1199810941.11.mhd",
      "5": "1.2.840.113704.1.111.4400.1131982359.11.mhd",
      "6": "1.3.12.2.1107.5.1.4.50585.4.0.7023259421321855.mhd",
      "7": "1.0.000.000000.0.00.0.0000000000.0000.0000000000.000.mhd",
      "8": "1.2.392.200036.9116.2.2.2.1762676169.1080882991.2256.mhd",
      "9": "2.16.840.1.113669.632.21.3825556854.538251028.390606191418956020.mhd"
    },
    "gt_fname": {
      "0": "1.2.840.113704.1.111.2296.1199810886.7.mhd",
      "1": "1.2.276.0.28.3.0.14.4.0.20090213134050413.mhd",
      "2": "1.2.276.0.28.3.0.14.4.0.20090213134114792.mhd",
      "3": "1.2.840.113704.1.111.2004.1131987870.11.mhd",
      "4": "1.2.840.113704.1.111.2296.1199810941.11.mhd",
      "5": "1.2.840.113704.1.111.4400.1131982359.11.mhd",
      "6": "1.3.12.2.1107.5.1.4.50585.4.0.7023259421321855.mhd",
      "7": "1.0.000.000000.0.00.0.0000000000.0000.0000000000.000.mhd",
      "8": "1.2.392.200036.9116.2.2.2.1762676169.1080882991.2256.mhd",
      "9": "2.16.840.1.113669.632.21.3825556854.538251028.390606191418956020.mhd"
    }
  },
```


(continues on next page)

(continued from previous page)

```
  "aggregates": {
    "dicecoefficient_mean": 0.6004146364647982,
    "dicecoefficient_std": 0.12096508479974993,
    "dicecoefficient_min": 0.32923801642370615,
    "dicecoefficient_max": 0.7250400040547097,
    "jaccardcoefficient_mean": 0.4378269970777743,
    "jaccardcoefficient_std": 0.11389145837530869,
    "jaccardcoefficient_min": 0.1970585994056237,
    "jaccardcoefficient_max": 0.5686766693547471,
  }
}
```

2.1.5 Evaluation Options

class grandchallenge.evaluation.models.**Config**(*id*, *created*, *modified*, *challenge*, *use_teams*, *score_title*, *score_jsonpath*, *score_error_jsonpath*, *score_default_sort*, *score_decimal_places*, *extra_results_columns*, *scoring_method_choice*, *result_display_choice*, *allow_submission_comments*, *display_submission_comments*, *supplementary_file_choice*, *supplementary_file_label*, *supplementary_file_help_text*, *show_supplementary_file_link*, *publication_url_choice*, *show_publication_url*, *daily_submission_limit*, *submission_page_html*, *auto_publish_new_results*, *display_all_metrics*, *submission_join_key*)

exception DoesNotExist

exception MultipleObjectsReturned

2.2 Template Tags

grandchallenge.evaluation.templatetags.evaluation_extras.**get_jsonpath**(*obj*, *jsonpath*)

Gets a value from a dictionary based on a jsonpath. It will only return one result, and if a key does not exist it will return an empty string as template tags should not raise errors.

Parameters

- **obj** (dict) – The dictionary to query
- **jsonpath** – The path to the object (singular)

Returns The most relevant object in the dictionary

CHAPTER 3

Processor specification

Version 1.0.1 - Date: 2019-03-29

Terms and definitions

This documents follows the definitions for keywords layed out in RFC 2119.

3.1 What is a processor?

A processor is a single or group of algorithms packaged in a docker container that can be applied to a set of files and produces some sort of output.

The processor container can be uploaded to the algorithms page. Then, it will be possible to upload some input files and the algorithm will be run on those files and produce an output.

3.2 Processor Inputs

Inputs to an algorithm are made a available to the processor in the directory `/input`. Valid inputs to an algorithm are files containing:

- MetaIO MHD/MHA files with accompanying zraw or raw files
- TIFF Images

The processor specification must describe which formats are supported.

3.2.1 MetaIO MHD/MHA files

MetaIO image files can be placed at any location of the `/input` directory. Accepted MetaIO image formats are mhd+raw, mhd+zraw, or mha files. MetaIO files in subdirectories may be ignored by processors. mhd and mha files must use the corresponding file extension to disambiguate between the two file formats.

3.2.2 Other files

Processors must be resilient with regards to extra files found in the `/input` directory. Any file that does not follow the specifications laid out above should be ignored by a processor. However, algorithms may specify in their documentation that additional "other files" are required for correct functioning of an algorithm. These additional files must be specified in the processor documentation.

3.3 Execution

A processor is a program that runs as main executable inside a Linux-based docker container implemented in any language. The algorithm must terminate by itself when it has finished processing data from the `/input` directory.

If an algorithm fails to process any items from the `/input` directory, it must terminate with and exitcode != 0. The algorithm should use stdout and stderr streams to explain the failure state. The contents of the `/output` directory shall be considered to be invalid in such a case. In all other cases, the algorithm must produce a valid `/output/results.json` file and finish with errorcode == 0.

If the algorithm fails to process some items from the `/input` directory, it must produce error messages for the given set of inputs as part of the results.json file that is described in the next section. Successful processing of input items must produce results that are written into the results.json for the given input items.

3.4 Processor Outputs

Outputs of an algorithm must be stored in the directory `/output/`. The output directory must consists of:

- A JSON file that contains the calculated results: `/output/results.json`
- Optional: Image files at user specified locations in the `/output/images/` directory.
  - Images must be saved using MetaIO mha or mhd files (either using compressed zraw or uncompressed raw binary blobs).
  - All images should be referred to in the `results.json` file. Images are referred to by adding a string-value in the `metrics` section denoting the relative path from the `results.json` to the image mhd or mha file. The semantics of the image are algorithm specific, meaning that images can contain any type information an algorithm author wants to store. Paths to image files should be prefixed with `filepath:`.

3.4.1 results.json

The `results.json` file must be a json file that contains a single array of *result objects*. Each input image results in one *result object*. A *result object* is a json-object that must contain computed algorithm results, references to related input files, and, in case a result cannot be computed, error messages explaining why the result cannot be computed. To encode these information, the result object for a given input item must use the following three keys:

- `entity`
- `metrics`
- `error_messages`

Result object sections

`entity`: This section links a result to a set of input files that were found in the input directory.

- Add suggestions for simple filename references
- Any json structure

`metrics`: The metrics section contains algorithm results. The results may be stored using any json structure, but the structure must be specified by the processor documentation. Also, all external files (except for `results.json` and `types.json`) must be listed in this section. Filepaths referring to external files should thereby be prefixed with the string `filepath:`. Unlisted external files may be ignored by frameworks running processors and not be copied to a persistent data store.

`error_messages`: A json list of human-readable strings denoting algorithm failures. If processor execution for a given set of input files was successful (see entity), this list must be empty. If the processing failed for a given set of inputs, at least one human readable error message denoting the failure state must be added to this list. In this case, the metrics section may be set to null or, if the metrics section is not set to null while errors are listed, it must be assumed that the metrics section is incomplete.:

```
[
  {
    "entity": ...,
    "metrics": ...,
    "error_messages": [
      ...
    ]
  },
  ...
]
```

3.5 Runtime requirements

Algorithms require system resources to run. The amount and type of system resources required to run a processor should be specified as docker labels. The following docker container labels should be used for specifying the required system resources are required to run a processor.

Docker container labels

Label	Values	Description
processor.cpus	Integer >= 1, Default: 1	The number of cpus the processor requires to finish computation in a reasonable amount of time
processor.cpu.capabilities	null or Stringlist	An optional list of processor capabilities that the used CPU must support to successfully run the processor. Can be an arbitrary list of flags, but at the moment of writing the following flags are supported: `avx`, `sse1`, `see2`, `sse3`, `sse4_1`, `sse4_2`, `mmx`
processor.memory	Size > 0, Default: 1G	The amount of memory to assign to the processor. This is the minimum amount of memory required with which the processor will successfully run.
processor.gpu_count	Integer >= 0 Default: 0	The number of CUDA-capable GPUs that are required to run the processor.
processor.gpu.compute_capability	null or Version, Default: null	Allows characterizing the required gpus in terms of supported CUDA compute capabilities. If specified, it must be a valid compute capability version.
processor.gpu.memory	null or Size, Default: null	The amount of gpu memory that must available on the type of graphics card that is made available to the container.

Value type descriptions

Type	Description
null	The string "null" (case insensitive). Represents none/nothing.
Integer	A whole number - no size limit. Valid examples: `-1`, `10`, `20222`, `4e1000`
Size	A size string. A size string consists of a positive Integer value combined with an optional size-character. Examples: `1000`, `5k`, `10G`, `100P` The size characters represent 1000-based unit prefixes for the unit "bytes". Size characters are case insensitive and the following associations are defined: k = kilo = 1000, g = giga = 1000^3, t = tera = 1000^4, p = peta = 1000^5, e = exa = 1000^6
Version	A version represents a version string. A version must start with at least one positive integer value. An arbitrary number of "."-separated additional positive integer values can follow. Examples: `3`, `3.2`, `0.0`, `3.0.0.0`, `0.1.0`
Stringlist	A comma-separated list of arbitrary strings. Strings cannot contain commas themselves: Example: `one,two,third string,four`

CHAPTER 4

Workstations

4.1 Models

Workstations are used to view, annotate and upload images to grand challenge. A *workstation admin* is able to upload a `WorkstationImage`, which is a docker container image. A `WorkstationImage` expose a http and, optionally, a websocket port. A *workstation user* can then launch a workstation `Session` for a particular `WorkstationImage`.

When a new session is started, a new container instance of the selected `WorkstationImage` is lauched on the docker host. The connection to the container will be proxied, and only accessible to the user that created the session. The proxy will map the http and websocket connections from the user to the running instance, which is mapped by the container hostname. The container instance will have the users API token set in the environment, so that it is able to interact with the grand challenge API as this user. The user is able to stop the container, otherwise it will be terminated after `maxmium_duration` is reached.

class `grandchallenge.workstations.models.`**Session**(**args*, ***kwargs*)

Tracks who has launched workstation images. The `WorkstationImage` will be launched as a `Service`. The `Session` is responsible for starting and stopping the `Service`.

Parameters

- **status** – Stores what has happened with the service, is it running, errored, etc?
- **creator** – Who created the session? This is also the only user that should be able to access the launched service.
- **workstation_image** – The container image that will be launched by this `Session`.
- **maximum_duration** – The maximum time that the service can be active before it is terminated
- **user_finished** – Indicates if the user has chosen to end the session early
- **history** – The history of this Session

exception DoesNotExist

exception MultipleObjectsReturned

environment

Return type dict

Returns *The environment variables that should be set on the container.*

expires_at

Return type datetime

Returns *The time when this session expires.*

hostname

Return type str

Returns *The unique hostname for this session*

save(**args*, ***kwargs*)
Save the session instance, starting or stopping the service if needed.

Return type None

save_without_historical_record(**args*, ***kwargs*)
Save model without saving a historical record

Make sure you know what you're doing before you use this method.

service

Return type Service

Returns *The service for this session, could be active or inactive.*

start()
Starts the service for this session, ensuring that the workstation_image is ready to be used and that WORKSTATIONS_MAXIMUM_SESSIONS has not been reached.

Raises ContainerExecException – If the service cannot be started.

Return type None

stop()
Stop the service for this session, cleaning up all of the containers.

Return type None

task_kwargs

Return type dict

Returns *The kwargs that need to be passed to celery to get this object*

update_status(*, *status*)
Updates the status of this session.

Parameters **status** (*((0, 'Queued'), (1, 'Started'), (2, 'Running'), (3, 'Failed'), (4, 'Stopped'))*) – The new status for this session.

Return type None

workstation_url

Return type str

Returns *The url that users will use to access the workstation instance.*

class grandchallenge.workstations.models.**Workstation**(**args*, ***kwargs*)
Store the title and description of a workstation.

exception DoesNotExist

exception MultipleObjectsReturned

latest_ready_image

Returns *The most recent container image for this workstation*

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class grandchallenge.workstations.models.**WorkstationImage**(**args*, ***kwargs*)
A `WorkstationImage` is a docker container image of a workstation.

Parameters

- **workstation** – A `Workstation` can have multiple `WorkstationImage`, that represent different versions of a workstation
- **http_port** – This container will expose a http server on this port
- **websocket_port** – This container will expose a websocket on this port. Any relative url that starts with `/mlab4d4c4142` will be proxied to this port.
- **initial_path** – The initial path that users will navigate to in order to load the workstation

exception DoesNotExist

exception MultipleObjectsReturned

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

grandchallenge.workstations.models.**delete_workstation_groups_hook**(*_, *instance*, *using*, ***__*)
Deletes the related groups.

We use a signal rather than overriding delete() to catch usages of bulk_delete.

CHAPTER 5

Reader Studies

A reader study enables you to have a set of readers answer a set of questions about a set of images.

Editors You can add multiple editors to your reader study. An editor is someone who can edit the reader study settings, add other editors, add and remove readers, add images and edit questions.

Readers A user who can read this study, creating an answer for each question and image in the study.

Cases The set of images that will be used in the study.

Hanging List How the each image will be presented to the user as a set of hanging protocols. For instance, you might want to present two images side by side and have a reader answer a question about both, or overlay one image on another.

5.1 Creating a Reader Study

A `ReaderStudy` can use any available `Workstation`. A `WorkstationConfig` can also be used for the study to customise the default appearance of the workstation.

5.2 Cases

Cases can be added to a reader study by adding `Image` instances. Multiple image formats are supported:

- `.mha`
- `.mhd` with the accompanying `.zraw` or `.raw` file
- `.tif`/`.tiff`
- `.jpg`/`.jpeg`
- `.png`
- 3D/4D DICOM support is also available, though this is experimental and not guaranteed to work on all `.dcm` images.

5.3 Defining the Hanging List

When you upload a set of images you have the option to automatically generate the default hanging list. The default hanging list presents each reader with 1 image per protocol.

You are able to customise the hanging list in the study edit page. Here, you are able to assign multiple images and overlays to each protocol. A `main` and `secondary` image port are available. Overlays can be applied to either image port by using the keys `main-overlay` and `secondary-overlay`.

5.4 Questions

A `Question` can be optional and the following `answer_type` options are available:

- Heading (not answerable)
- Bool
- Single line text
- Multiline text

The following annotation answer types are also available:

- Distance measurement
- Multiple distance measurements
- 2D bounding box

To use an annotation answer type you must also select the image port where the annotation will be made.

5.5 Adding Ground Truth

To monitor the performance of the readers you are able add ground truth to a reader study by uploading a csv file.

If ground truth has been added to a `ReaderStudy`, any `Answer` given by a reader is evaluated by applying the `scoring_function` chosen for the `Question`.

The scores can then be compared on the `leaderboard`. Statistics are also available based on these scores: the average and total scores for each question as well as for each case are displayed in the `statistics` view.

`grandchallenge.reader_studies.models.`**`ANSWER_TYPE_SCHEMA = {'$schema': 'http://json-schema`**
Schema used to validate if answers are of the correct type and format.

class `grandchallenge.reader_studies.models.`**`Answer`**(**args*, ***kwargs*)
An `Answer` can be provided to a `Question` that is a part of a `ReaderStudy`.

exception DoesNotExist

exception MultipleObjectsReturned

api_url
API url for this `Answer`.

calculate_score(*ground_truth*)
Calculate the score for this `Answer` based on `ground_truth`.

csv_values
Values that are included in this `Answer`'s csv export.

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

static validate(***, *creator*, *question*, *answer*, *images*, *is_ground_truth=False*)
Validates all fields provided for `answer`.

class grandchallenge.reader_studies.models.**Question**(*id*, *created*, *modified*, *reader_study*, *question_text*, *help_text*, *answer_type*, *image_port*, *required*, *direction*, *scoring_function*, *order*)

exception DoesNotExist

exception MultipleObjectsReturned

api_url
API url for this `Question`.

calculate_score(*answer*, *ground_truth*)
Calculates the score for `answer` by applying `scoring_function` to `answer` and `ground_truth`.

clean()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

csv_values
Values that are included in this `Question`'s csv export.

is_answer_valid(***, *answer*)
Validates `answer` against `ANSWER_TYPE_SCHEMA`.

is_fully_editable
`True` if no `Answer` has been given for this `Question`.

read_only_fields
`question_text`, `answer_type`, `image_port`, `required` if this `Question` is fully editable, an empty list otherwise.

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class grandchallenge.reader_studies.models.**ReaderStudy**(**args*, ***kwargs*)
Reader Study model.

A reader study is a tool that allows users to have a set of readers answer a set of questions on a set of images (cases).

exception DoesNotExist

exception MultipleObjectsReturned

add_editor(*user*)
Adds `user` as an editor for this `ReaderStudy`.

add_ground_truth(***, *data*, *user*)
Add ground truth answers provided by `data` for this `ReaderStudy`.

add_reader(*user*)
Adds `user` as a reader for this `ReaderStudy`.

answerable_question_count
The number of answerable questions for this `ReaderStudy`.

answerable_questions
All questions for this `ReaderStudy` except those with answer type *heading*.

generate_hanging_list()
Generates a new hanging list.

Each image in the `ReaderStudy` is assigned to the primary port of its own hanging.

get_hanging_list_images_for_user(*, *user*)
Returns a shuffled list of the hanging list images for a particular user.

The shuffle is seeded with the users pk, and using `RandomState` from numpy guarantees that the ordering will be consistent across python/library versions. Returns the normal list if `shuffle_hanging_list` is `False`.

get_progress_for_user(*user*)
Returns the percentage of completed hangings and questions for `user`.

hanging_image_names
Names for all images in the hanging list.

hanging_list_diff
Returns the diff between the images added to the study and the images in the hanging list.

hanging_list_images
Substitutes the image name for the image detail api url for each image defined in the hanging list.

hanging_list_valid
Tests that all of the study images are included in the hanging list exactly once.

image_groups
Names of the images as they are grouped in the hanging list.

is_editor(*user*)
Checks if `user` is an editor for this `ReaderStudy`.

is_reader(*user*)
Checks if `user` is a reader for this `ReaderStudy`.

is_valid
Returns `True` if the hanging list is valid and there are no duplicate image names in this `ReaderStudy` and `False` otherwise.

leaderboard
The leaderboard for this `ReaderStudy`.

non_unique_study_image_names
Returns all of the non-unique image names for this `ReaderStudy`.

remove_editor(*user*)
Removes `user` as an editor for this `ReaderStudy`.

remove_reader(*user*)
Removes `user` as a reader for this `ReaderStudy`.

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

score_for_user(*user*)
Returns the average and total score for answers given by `user`.

scores_by_user
The average and total scores for this `ReaderStudy` grouped by user.

statistics
Statistics per question and case based on the total / average score.

study_image_names
Names for all images added to this `ReaderStudy`.

`grandchallenge.reader_studies.models.`**delete_reader_study_groups_hook**(*_, *instance*, *using*, **__)

Deletes the related groups.

We use a signal rather than overriding delete() to catch usages of bulk_delete.

CHAPTER 6

Design decisions

This section contains extra information about design decisions that have been made during the development of grand-challenge.org. This section is intended for software developers who want to contribute to the codebase. It is a summary of large scale decisions made by the development team and will help (we hope) to guide future developments so that all the overall design of grand-challenge remains consistent.

6.1 Definitions

This document uses sentences written in *italic text* to denote design decisions that were decided upon during previous design meetings. These are choices that should discussed within the development team should it become necessary to diverege from them.

6.2 Image database objects

The app "cases" contains two types of database objects to store a single image in database:

- `cases.models.Image`
- `cases.models.ImageFile`

In the current design, the Image-object denotes a container object for an image. This image can have one or more *representations*. A representation consists of one or more files with a given data type, which are stored as ImageFile objects that belong to a given Image object. A data type might thereby consist of more than one file, like for example is the case for multislice-dicom files. Therefore, an application must enumerate all ImageFiles with a given type to access all the full image data that uses the given data type. It can safely be assumed that:

- *An Image will only include a single set of ImageFile objects for a given data type.*
- *The data type is stored for each ImageFile in the image_type field.*

Abstract, descriptive information about an image can typically be queried on the Image object directly. Examples are: resolution of the image, color space, modality, and more. However, in the future it might become necessary to move

some of these descriptors to the ImageFile-level, in case we include data types that constrain the possible values of any of these descriptors. Therefore:

- *Generally, we should not include image types in grand-challenge that limit the choice of values for any of the Image-object descriptor fields.*

However, in case, we ever need to include a restrictive data representation to our database:

- *We will move Image-level descriptive fields to ImageFile-fields.*
- *This will be decided upon on a case-by-case basis.*
- *Only required fields will be moved (most descriptive fields should remain on the Image object).*

At the moment of writing (2019-08) all image types that are supported by grand-challenge can encode images with any combination of Image-fields. So data type compatibility between different image representations is currently not an issue.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

grandchallenge.evaluation.templatetags.evaluation_extras, 9
grandchallenge.reader_studies.models, 19
grandchallenge.workstations.models, 15

Index

A

add_editor() (*grandchallenge.reader_studies.models.ReaderStudy method*), 21
add_ground_truth() (*grandchallenge.reader_studies.models.ReaderStudy method*), 21
add_reader() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
Answer (*class in grandchallenge.reader_studies.models*), 20
Answer.DoesNotExist, 20
Answer.MultipleObjectsReturned, 20
ANSWER_TYPE_SCHEMA (*in module grandchallenge.reader_studies.models*), 20
answerable_question_count (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22
answerable_questions (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22
api_url (*grandchallenge.reader_studies.models.Answer attribute*), 20
api_url (*grandchallenge.reader_studies.models.Question attribute*), 21

C

calculate_score() (*grandchallenge.reader_studies.models.Answer method*), 20
calculate_score() (*grandchallenge.reader_studies.models.Question method*), 21
clean() (*grandchallenge.reader_studies.models.Question method*), 21
Config (*class in grandchallenge.evaluation.models*), 9
Config.DoesNotExist, 9
Config.MultipleObjectsReturned, 9
csv_values (*grandchallenge.reader_studies.models.Answer attribute*), 20
csv_values (*grandchallenge.reader_studies.models.Question attribute*), 21

D

delete_reader_study_groups_hook() (*in module grandchallenge.reader_studies.models*), 23
delete_workstation_groups_hook() (*in module grandchallenge.workstations.models*), 17

E

environment (*grandchallenge.workstations.models.Session attribute*), 15
expires_at (*grandchallenge.workstations.models.Session attribute*), 16

G

generate_hanging_list() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
get_hanging_list_images_for_user() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
get_jsonpath() (*in module grandchallenge.evaluation.templatetags.evaluation_extras*), 9
get_progress_for_user() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
grandchallenge.evaluation.templatetags.evaluation_e (*module*), 9
grandchallenge.reader_studies.models (*module*), 19

grandchallenge.workstations.models (*module*), 15

H

hanging_image_names (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22
hanging_list_diff (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22
hanging_list_images (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22
hanging_list_valid (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22
hostname (*grandchallenge.workstations.models.Session attribute*), 16

I

image_groups (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22
is_answer_valid() (*grandchallenge.reader_studies.models.Question method*), 21
is_editor() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
is_fully_editable (*grandchallenge.reader_studies.models.Question attribute*), 21
is_reader() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
is_valid (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22

L

latest_ready_image (*grandchallenge.workstations.models.Workstation attribute*), 17
leaderboard (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22

N

non_unique_study_image_names (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 22

Q

Question (*class in grandchallenge.reader_studies.models*), 21
Question.DoesNotExist, 21
Question.MultipleObjectsReturned, 21

R

read_only_fields (*grandchallenge.reader_studies.models.Question attribute*), 21
ReaderStudy (*class in grandchallenge.reader_studies.models*), 21
ReaderStudy.DoesNotExist, 21
ReaderStudy.MultipleObjectsReturned, 21
remove_editor() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
remove_reader() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22

S

save() (*grandchallenge.reader_studies.models.Answer method*), 20
save() (*grandchallenge.reader_studies.models.Question method*), 21
save() (*grandchallenge.reader_studies.models.ReaderStudy method*), 22
save() (*grandchallenge.workstations.models.Session method*), 16
save() (*grandchallenge.workstations.models.Workstation method*), 17
save() (*grandchallenge.workstations.models.WorkstationImage method*), 17
save_without_historical_record() (*grandchallenge.workstations.models.Session method*), 16
score_for_user() (*grandchallenge.reader_studies.models.ReaderStudy method*), 23
scores_by_user (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 23
service (*grandchallenge.workstations.models.Session attribute*), 16
Session (*class in grandchallenge.workstations.models*), 15
Session.DoesNotExist, 15
Session.MultipleObjectsReturned, 15
start() (*grandchallenge.workstations.models.Session method*), 16
statistics (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 23

stop() (*grandchallenge.workstations.models.Session method*), 16
study_image_names (*grandchallenge.reader_studies.models.ReaderStudy attribute*), 23

T

task_kwargs (*grandchallenge.workstations.models.Session attribute*), 16

U

update_status() (*grandchallenge.workstations.models.Session method*), 16

V

validate() (*grandchallenge.reader_studies.models.Answer static method*), 21

W

Workstation (*class in grandchallenge.workstations.models*), 16
Workstation.DoesNotExist, 16
Workstation.MultipleObjectsReturned, 17
workstation_url (*grandchallenge.workstations.models.Session attribute*), 16
WorkstationImage (*class in grandchallenge.workstations.models*), 17
WorkstationImage.DoesNotExist, 17
WorkstationImage.MultipleObjectsReturned, 17