edx-submissions Documentation

Release 0.0.1

edX

September 21, 2016

Contents

`submissions` is a Django app that defines a common interface for creating submissions and scores.

CHAPTER 1

Setup

To install the `submissions` app:

```
python setup.py install
```

CHAPTER 2

API Documentation

2.1 Public API

Every Django application in edx-ora2 has an *api.py* that is its public interface. If you are using one of these applications from the outside, you should only import things from that module. The ground rules for api modules are:

1. All inputs and outputs must be trivially serializable to JSON. This means *None*, *int*, *float*, *unicode*, *list*, *tuple*, *dict*, and *datetime*.
2. Returned objects should not have methods or business logic attached to them.
3. Caller should assume that these calls can be moderately expensive, as they may one day move out of process and become network calls. So calling something a hundred times in a loop should be avoided.

2.1.1 Submissions

Public interface for the submissions app.

exception `submissions.api.`**`SubmissionError`**
An error that occurs during submission actions.

This error is raised when the submission API cannot perform a requested action.

exception `submissions.api.`**`SubmissionInternalError`**
An error internal to the Submission API has occurred.

This error is raised when an error occurs that is not caused by incorrect use of the API, but rather internal implementation of the underlying services.

exception `submissions.api.`**`SubmissionNotFoundError`**
This error is raised when no submission is found for the request.

If a state is specified in a call to the API that results in no matching Submissions, this error may be raised.

exception `submissions.api.`**`SubmissionRequestError`**(*msg=''*, *field_errors=None*)
This error is raised when there was a request-specific error

This error is reserved for problems specific to the use of the API.

`submissions.api.`**`create_submission`**(*student_item_dict*, *answer*, *submitted_at=None*, *attempt_number=None*)
Creates a submission for assessment.

Generic means by which to submit an answer for assessment.

Args:

student_item_dict (dict): The student_item this submission is associated with. This is used to determine which course, student, and location this submission belongs to.

answer (JSON-serializable): The answer given by the student to be assessed.

submitted_at (datetime): The date in which this submission was submitted. If not specified, defaults to the current date.

attempt_number (int): A student may be able to submit multiple attempts per question. This allows the designated attempt to be overridden. If the attempt is not specified, it will take the most recent submission, as specified by the submitted_at time, and use its attempt_number plus one.

Returns: dict: A representation of the created Submission. The submission contains five attributes: student_item, attempt_number, submitted_at, created_at, and answer. 'student_item' is the ID of the related student item for the submission. 'attempt_number' is the attempt this submission represents for this question. 'submitted_at' represents the time this submission was submitted, which can be configured, versus the 'created_at' date, which is when the submission is first created.

Raises:

SubmissionRequestError: Raised when there are validation errors for the student item or submission. This can be caused by the student item missing required values, the submission being too long, the attempt_number is negative, or the given submitted_at time is invalid.

SubmissionInternalError: Raised when submission access causes an internal error.

Examples:

```
>>> student_item_dict = dict(
>>>     student_id="Tim",
>>>     item_id="item_1",
>>>     course_id="course_1",
>>>     item_type="type_one"
>>> )
>>> create_submission(student_item_dict, "The answer is 42.", datetime.utcnow, 1)
{
    'student_item': 2,
    'attempt_number': 1,
    'submitted_at': datetime.datetime(2014, 1, 29, 17, 14, 52, 649284 tzinfo=<UTC>),
    'created_at': datetime.datetime(2014, 1, 29, 17, 14, 52, 668850, tzinfo=<UTC>),
    'answer': u'The answer is 42.'
}
```

submissions.api.**get_all_course_submission_information**(*course_id*, *item_type*, *read_replica=True*)

For the given course, get all student items of the given item type, all the submissions for those itemes, and the latest scores for each item. If a submission was given a score that is not the latest score for the relevant student item, it will still be included but without score.

Args: course_id (str): The course that we are getting submissions from. item_type (str): The type of items that we are getting submissions for. read_replica (bool): Try to use the database's read replica if it's available.

Yields: A tuple of three dictionaries representing: (1) a student item with the following fields:

student_id course_id student_item item_type

2. a submission with the following fields: student_item attempt_number submitted_at created_at answer
3. a score with the following fields, if one exists and it is the latest score: (if both conditions are not met, an empty dict is returned here) student_item submission points_earned points_possible created_at submission_uuid

submissions.api.**get_all_submissions**(*course_id*, *item_id*, *item_type*, *read_replica=True*)
For the given item, get the most recent submission for every student who has submitted.

This may return a very large result set! It is implemented as a generator for efficiency.

Args:

course_id, item_id, item_type (string): The values of the respective student_item fields to filter the submissions by.

read_replica (bool): If true, attempt to use the read replica database. If no read replica is available, use the default database.

Yields:

Dicts representing the submissions with the following fields: student_item student_id attempt_number submitted_at created_at answer

Raises: Cannot fail unless there's a database error, but may return an empty iterable.

submissions.api.**get_latest_score_for_submission**(*submission_uuid*, *read_replica=False*)
Retrieve the latest score for a particular submission.

Args: submission_uuid (str): The UUID of the submission to retrieve.

Kwargs:

read_replica (bool): If true, attempt to use the read replica database. If no read replica is available, use the default database.

Returns: dict: The serialized score model, or None if no score is available.

submissions.api.**get_score**(*student_item*)
Get the score for a particular student item

Each student item should have a unique score. This function will return the score if it is available. A score is only calculated for a student item if it has completed the workflow for a particular assessment module.

Args:

student_item (dict): The dictionary representation of a student item. Function returns the score related to this student item.

Returns:

score (dict): The score associated with this student item. None if there is no score found.

Raises:

SubmissionInternalError: Raised if a score cannot be retrieved because of an internal server error.

Examples:

```
>>> student_item = {
>>>     "student_id":"Tim",
>>>     "course_id":"TestCourse",
>>>     "item_id":"u_67",
>>>     "item_type":"openassessment"
>>> }
>>>
>>> get_score(student_item)
[{
    'student_item': 2,
    'submission': 2,
    'points_earned': 8,
```

```
            'points_possible': 20,
            'created_at': datetime.datetime(2014, 2, 7, 18, 30, 1, 807911, tzinfo=<UTC>)
        }]
```

submissions.api.**get_scores**(*course_id*, *student_id*)

Return a dict mapping item_ids -> (points_earned, points_possible).

This method would be used by an LMS to find all the scores for a given student in a given course.

Scores that are "hidden" (because they have points earned set to zero) are excluded from the results.

Args: course_id (str): Course ID, used to do a lookup on the *StudentItem*. student_id (str): Student ID, used to do a lookup on the *StudentItem*.

Returns: dict: The keys are *item_id's ('str*) and the values are tuples of *(points_earned, points_possible)*. All points are integer values and represent the raw, unweighted scores. Submissions does not have any concept of weights. If there are no entries matching the *course_id* or *student_id*, we simply return an empty dictionary. This is not considered an error because there might be many queries for the progress page of a person who has never submitted anything.

Raises: SubmissionInternalError: An unexpected error occurred while resetting scores.

submissions.api.**get_submission**(*submission_uuid*, *read_replica=False*)

Retrieves a single submission by uuid.

Args: submission_uuid (str): Identifier for the submission.

Kwargs:

read_replica (bool): If true, attempt to use the read replica database. If no read replica is available, use the default database.

Raises: SubmissionNotFoundError: Raised if the submission does not exist. SubmissionRequestError: Raised if the search parameter is not a string. SubmissionInternalError: Raised for unknown errors.

Examples:

```
        >>> get_submission("20b78e0f32df805d21064fc912f40e9ae5ab260d")
        {
            'student_item': 2,
            'attempt_number': 1,
            'submitted_at': datetime.datetime(2014, 1, 29, 23, 14, 52, 649284, tzinfo=<UTC>),
            'created_at': datetime.datetime(2014, 1, 29, 17, 14, 52, 668850, tzinfo=<UTC>),
            'answer': u'The answer is 42.'
        }
```

submissions.api.**get_submission_and_student**(*uuid*, *read_replica=False*)

Retrieve a submission by its unique identifier, including the associated student item.

Args: uuid (str): the unique identifier of the submission.

Kwargs:

read_replica (bool): If true, attempt to use the read replica database. If no read replica is available, use the default database.

Returns: Serialized Submission model (dict) containing a serialized StudentItem model

Raises: SubmissionNotFoundError: Raised if the submission does not exist. SubmissionRequestError: Raised if the search parameter is not a string. SubmissionInternalError: Raised for unknown errors.

submissions.api.**get_submissions**(*student_item_dict*, *limit=None*)

Retrieves the submissions for the specified student item, ordered by most recent submitted date.

Returns the submissions relative to the specified student item. Exception thrown if no submission is found relative to this location.

Args:

student_item_dict (dict): The location of the problem this submission is associated with, as defined by a course, student, and item.

limit (int): Optional parameter for limiting the returned number of submissions associated with this student item. If not specified, all associated submissions are returned.

Returns: List dict: A list of dicts for the associated student item. The submission contains five attributes: student_item, attempt_number, submitted_at, created_at, and answer. 'student_item' is the ID of the related student item for the submission. 'attempt_number' is the attempt this submission represents for this question. 'submitted_at' represents the time this submission was submitted, which can be configured, versus the 'created_at' date, which is when the submission is first created.

Raises:

SubmissionRequestError: Raised when the associated student item fails validation.

SubmissionNotFoundError: Raised when a submission cannot be found for the associated student item.

Examples:

```
>>> student_item_dict = dict(
>>>     student_id="Tim",
>>>     item_id="item_1",
>>>     course_id="course_1",
>>>     item_type="type_one"
>>> )
>>> get_submissions(student_item_dict, 3)
[{
    'student_item': 2,
    'attempt_number': 1,
    'submitted_at': datetime.datetime(2014, 1, 29, 23, 14, 52, 649284, tzinfo=<UTC>),
    'created_at': datetime.datetime(2014, 1, 29, 17, 14, 52, 668850, tzinfo=<UTC>),
    'answer': u'The answer is 42.'
}]
```

submissions.api.**get_top_submissions**(*course_id*, *item_id*, *item_type*, *number_of_top_scores*, *use_cache=True*, *read_replica=True*)

Get a number of top scores for an assessment based on a particular student item

This function will return top scores for the piece of assessment. It will consider only the latest and greater than 0 score for a piece of assessment. A score is only calculated for a student item if it has completed the workflow for a particular assessment module.

In general, users of top submissions can tolerate some latency in the search results, so by default this call uses a cache and the read replica (if available).

Args: course_id (str): The course to retrieve for the top scores item_id (str): The item within the course to retrieve for the top scores item_type (str): The type of item to retrieve number_of_top_scores (int): The number of scores to return, greater than 0 and no more than 100.

Kwargs: use_cache (bool): If true, check the cache before retrieving querying the database. read_replica (bool): If true, attempt to use the read replica database.

If no read replica is available, use the default database.

Returns:

topscores (dict): The top scores for the assessment for the student item. An empty array if there are no scores or all scores are 0.

Raises:

SubmissionNotFoundError: Raised when a submission cannot be found for the associated student item.

SubmissionRequestError: Raised when the number of top scores is higher than the MAX_TOP_SUBMISSIONS constant.

Examples:

```
>>> course_id = "TestCourse"
>>> item_id = "u_67"
>>> item_type = "openassessment"
>>> number_of_top_scores = 10
>>>
>>> get_top_submissions(course_id, item_id, item_type, number_of_top_scores)
[{
    'score': 20,
    'content': "Platypus"
},{
    'score': 16,
    'content': "Frog"
}]
```

submissions.api.**reset_score**(*student_id*, *course_id*, *item_id*, *clear_state=False*)

Reset scores for a specific student on a specific problem.

Note: this does *not* delete *Score* models from the database, since these are immutable. It simply creates a new score with the "reset" flag set to True.

Args: student_id (unicode): The ID of the student for whom to reset scores. course_id (unicode): The ID of the course containing the item to reset. item_id (unicode): The ID of the item for which to reset scores. clear_state (bool): If True, will appear to delete any submissions associated with the specified StudentItem

Returns: None

Raises: SubmissionInternalError: An unexpected error occurred while resetting scores.

submissions.api.**set_score**(*submission_uuid*, *points_earned*, *points_possible*, *annotation_creator=None*, *annotation_type=None*, *annotation_reason=None*)

Set a score for a particular submission.

Sets the score for a particular submission. This score is calculated externally to the API.

Args: submission_uuid (str): UUID for the submission (must exist). points_earned (int): The earned points for this submission. points_possible (int): The total points possible for this particular student item.

annotation_creator (str): An optional field for recording who gave this particular score annotation_type (str): An optional field for recording what type of annotation should be created,

e.g. "staff_override".

annotation_reason (str): An optional field for recording why this score was set to its value.

Returns: None

Raises:

SubmissionInternalError: Thrown if there was an internal error while attempting to save the score.

SubmissionRequestError: Thrown if the given student item or submission are not found.

Examples:

```
>>> set_score("a778b933-9fb3-11e3-9c0f-040ccee02800", 11, 12)
{
    'student_item': 2,
    'submission': 1,
    'points_earned': 11,
    'points_possible': 12,
    'created_at': datetime.datetime(2014, 2, 7, 20, 6, 42, 331156, tzinfo=<UTC>)
}
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

S

C

G

R

S