
LIVE Documentation

Release 3.x

Josef Wachtler

May 12, 2023

CONTENTS

1	Overview	1
2	Selected Topics	3
2.1	Install and Run	3
2.1.1	Use vagrant/dev	3
2.1.2	Install local	3
2.1.3	(Re-)Init a Database	4
2.1.4	Build the Documentation	4
2.2	Translation	4
2.3	Config Framework	5
2.3.1	Storing Config Entries	5
2.3.2	Accessing Config Entries	6
2.4	Analysis Framework	6
2.4.1	Computing the Attention Level	6
2.4.2	Register Functions	7
2.5	Interaction Framework	8
2.5.1	General Steps	8
2.5.2	Models	9
2.5.3	Views and Templates	9
3	Publications	23
4	Source Code Documentation	25
4.1	Manage-Files	25
4.1.1	manage.py	25
4.1.2	manage_vagrant_lampy.py	25
4.1.3	manage_production_dev.py	25
4.1.4	manage_production.py	25
4.2	Package: core	26
4.2.1	Module: core.admin	26
4.2.2	Module: core.apps	27
4.2.3	Module: core.config	27
4.2.4	Module: core.context_processors	28
4.2.5	Module: core.fields	28
4.2.6	Module: core.forms	30
4.2.7	Module: core.middleware	33
4.2.8	Module: core.models	33
4.2.9	Module: core.research	42
4.2.10	Package: core.templatetags	44
4.2.11	Module: core.tests	45
4.2.12	Module: core.translation	46
4.2.13	Module: core.urls	46
4.2.14	Module: core.utils	46
4.2.15	Module: core.validators	47

4.2.16	Package: core.views	48
4.3	Package: LIVE	59
4.3.1	Module: LIVE.settings	59
4.3.2	Module: LIVE.settings_production	59
4.3.3	Module: LIVE.settings_production_dev	59
4.3.4	Module: LIVE.settings_vagrant_lampy	59
4.3.5	Module: LIVE.urls	59
4.4	Package: questionpool	59
4.4.1	Module: questionpool.admin	59
4.4.2	Module: questionpool.apps	60
4.4.3	Module: questionpool.forms	60
4.4.4	Module: questionpool.models	62
4.4.5	Module: questionpool.tests	63
4.4.6	Module: questionpool.urls	63
4.4.7	Module: questionpool.views	63
4.5	Package: interactions	66
4.5.1	Module: interactions.admin	67
4.5.2	Module: interactions.analysis	67
4.5.3	Module: interactions.apps	68
4.5.4	Module: interactions.forms	68
4.5.5	Module: interactions.models	68
4.5.6	Module: interactions.student	72
4.5.7	Module: interactions.teacher	73
4.5.8	Module: interactions.tests	74
4.5.9	Module: interactions.translation	74
4.5.10	Module: interactions.urls	74
4.5.11	Module: interactions.utils	74
4.5.12	Module: interactions.views	75
4.5.13	Package: interactions.ask_students	77
4.5.14	Package: interactions.ask_teacher	80
4.5.15	Package: interactions.attention	84
4.5.16	Package: interactions.planned_questions	85
4.5.17	Package: interactions.planned_text_questions	92
4.5.18	Package: interactions.planned_programming_tasks	98
4.5.19	Package: interactions.report_problem	105
4.5.20	Package: interactions.simple	105
4.5.21	Package: interactions.solve_captcha	107
4.6	Package: research	108
4.6.1	Module: research.admin	108
4.6.2	Module: research.apps	108
4.6.3	Module: research.models	109
4.6.4	Module: research.tests	109
4.6.5	Module: research.urls	109
4.6.6	Module: research.utils	109
4.6.7	Module: research.views	109

Python Module Index	121
----------------------------	------------

Index	123
--------------	------------

OVERVIEW

This is the documentation of LIVE (short for Live Interaction in Virtual Learning Environments). It is a web-platform which enriches videos or live-broadcastings with different methods of interactivity. From a technical point of view LIVE is based on python, django and some other libraries.

On top of this framework there is the project LIVE which consists of the following django apps:

- core: contains all basic functionalities
- research: enables the download of all data as spreadsheets for research purposes
- questionpool: offers pools of questions which could be used at the interaction methods
- interactions: provides a framework for implementing the different interaction methods which are sub apps of this app

Currently the following interaction methods are available:

- **Simple Questions (*interactions.simple*)**
 - general questions which are not related to the content of the video
 - random and automatic
 - useful for bridging a longer phase during which no content-related questions are asked
- **Solve CAPTCHAs (*interactions.solve_captcha*)**
 - used for the same reasons as the simple questions above
- **Text-Based Questions**
 - the teacher can ask text-based questions to the students
 - at live-broadcastings (*interactions.ask_students*), she/he could enter the question into a textbox and send it to the students instantly
 - in videos (*interactions.planned_text_questions*), she/he must place such a question at a specific position in the video before releasing it
- **Multiple-Choice Questions (*interactions.planned_questions*)**
 - real multiple-choice questions or true/false questions
 - before deploying the video, the teacher could add questions of this type at pre-defined timepoints throughout the video
- **Programming Tasks (*interactions.planned_programming_tasks*)**
 - little programming tasks which are executed and the output is compared to a reference
 - before deploying the video, the teacher could add questions of this type at pre-defined timepoints throughout the video
- **Ask Teacher (*interactions.ask_teacher*)**
 - the attendees can ask a question to the teacher by entering it in a textbox
 - to answer the question, the teacher could use a specific dialog or she/he could send it by e-mail

- **Report a Technical Problem** (*interactions.report_problem*)
 - the students can report a technical problem to the teacher via a dialog
 - this feature is mainly used during live-broadcastings to report problems with the video feed
- **Set Attention** (*interactions.attention*)
 - with the help of a slider, the watchers can indicate their current level of attention

SELECTED TOPICS

2.1 Install and Run

Use one of the following possibilities to create a development environment for LIVE in which it can run.

2.1.1 Use vagrant/dev

on the host run:

```
1 git clone git@gitlab.tugraz.at:llt/learninglab_coop/live/project_live.git
2 cd project_live/vagrant/dev
3 vagrant up
4 vagrant ssh
```

in the box run:

```
1 cd live/env/
2 source bin/activate
3 cd LIVE/
4 python manage.py runserver 0.0.0.0:8080
```

Now you can access LIVE at <http://127.0.0.1:8080/>

2.1.2 Install local

This inits a virtual environment and runs the integrated dev-server with a preconfigured sqlite db under ubuntu 20.04:

Install pre-requirements:

```
1 sudo apt-get install python3-dev python3-pip libfreetype6-dev libjpeg-dev zlib1g-dev
```

Create virtual environment or use <http://virtualenvwrapper.readthedocs.io/en/stable/> (recommended):

```
1 python3 -m venv py3venv --without-pip
2 cd py3venv
3 source bin/activate
4 wget https://bootstrap.pypa.io/get-pip.py
5 python get-pip.py
6 deactivate
7 source bin/activate
```

Get LIVE:

```
1 git clone git@gitlab.tugraz.at:llt/learninglab_coop/live/project_live.git
2 cd project_live/
```

Install requirements:

```
1 pip install -r requirements.txt
```

Run dev-server:

```
1 cd LIVE
2 python manage.py runserver
```

Now you can access LIVE at <http://127.0.0.1:8000/>

2.1.3 (Re-)Init a Database

```
1 rm dev-data/live.db
2 cd LIVE
3
4 python manage.py makemigrations
5 python manage.py migrate
6 python manage.py loaddata settings
7 python manage.py loaddata user
```

2.1.4 Build the Documentation

To build the documentation run the following commands:

```
1 cd doc/src
2
3 make html
4 make latexpdf
```

Now the documentation can be found in doc/_build.

2.2 Translation

LIVE is implemented fully ready for translation. The base language is English and currently there is a German translation. For that the typical mechanism of django is used.

Because of the fact that the interaction methods are placed inside the interactions app a special way of calling the making and compiling of messages is required. This means that they are ignored and so the commands for the interactions app looks like the following:

```
1 cd interactions && django-admin.py makemessages -l de \
2   -i ask_teacher \
3   -i ask_students \
4   -i attention \
5   -i planed_questions \
6   -i report_problem \
7   -i simple \
8   -i solve_captcha \
9   -i planed_text_questions \
10  -i planed_progamming_tasks
```

To create the translations for the whole project the commands are encapsulated in a script:

```
1 ./messages.sh make
2 ./messages.sh compile
```

2.3 Config Framework

The config framework provides a mechanism to store key/value pairs in the database. Furthermore there are some functions to access the values by their keys.

2.3.1 Storing Config Entries

The key/value pairs are stored in `core.models.ConfigEntry`. This model provides the following fields:

- key: the key to identify the config value in uppercase
- value: the config value
- interaction_method: the id of the related interaction method or “null” for a global config entry. See “interactions/fixtures/settings.json” for the value.

The entries should be listed as a fixture in the directory “fixtures” of a django app. The name of the fixture should be “settings.json” for default settings. These settings could be overwritten for productive usage with a fixture named “settings_production.json”.

The following example shows such a fixture:

```
1 [
2   {
3     "pk": 1101,
4     "model": "core.configentry",
5     "fields": {
6       "interaction_method": 1,
7       "value": "20",
8       "key": "TRIGGERS_PER_HOUR"
9     }
10  },
11 ]
```

The field “pk” has to be a unique id. It is recommended to use the following schema:

- 4 digits
- The first digit is the id/pk of the interaction type. See “interactions/fixtures/settings.json” for the value.
- The second digit is the id/pk of the interaction method. See “interactions/fixtures/settings.json” for the value.
- The last two digits are a growing number.

For all the other fields the example should be followed.

2.3.2 Accessing Config Entries

There are four helper functions to get a config value by its key in `core.config`. They offer the possibility to get a string, an int, a bool and bytes. Each function has the following keyword arguments:

- `key`: the key of the value
- `interaction_method`: an instance of `interactions.models.InteractionMethod` representing the related interaction method or `None` for a global config value

2.4 Analysis Framework

The analysis framework is responsible for providing a detailed analysis of the watching users. It consists of two parts:

- A detailed recording of the joined and watched timespans of each user. This is implemented in the core package and it is not required to handle it at the interaction methods.
- The computation of an attention level which is a number ranging from 0 (very absent) to 100 (very attentive). It has to be computed by each interaction method.

For more information read this publication:

- Wachtler, J., & Ebner, M. (2014b). Attention Profiling Algorithm for Video-Based Lectures. In Learning and Collaboration Technologies. Designing and Developing Novel Learning Experiences (pp. 358-367). Springer International Publishing.

2.4.1 Computing the Attention Level

Each interaction method has to offer functions to compute the attention level.

The first function has to compute the attention level for a users `core.models.History` which means that this is the computation after watching. This function has to look like the following example:

```
1 def get_attention_level_per_history(history):
2     """
3     This function returns the attention of a history of a joined user.
4
5     Keyword arguments:
6     - history -- the history of a joined user
7     """
8     try:
9         im = InteractionMethod.objects.get(
10             title_en='Interaction method name'
11         )
12     except InteractionMethod.DoesNotExist:
13         return -1
14
15     a = compute() # compute the attention level
16     return int(a)
```

The second function has to compute the attention level while watching. This function has to look like the following example:

```
1 def get_attention_level(joineduser, at_relative, at_absolute):
2     """
3     This function returns a timebased attention level for a joined user
4     during an event.
```

(continues on next page)

(continued from previous page)

```

5
6     Keyword arguments:
7         - joineduser -- the joined user
8         - at_relative -- current position
9         - at_absolute -- current time
10
11     """
12     try:
13         im = InteractionMethod.objects.get(
14             title_en='Interaction method name'
15         )
16     except InteractionMethod.DoesNotExist:
17         return -1
18
19     a = compute() # compute the attention level
20     return int(a)

```

Because in most of the cases the attention level is based on the reaction time to interactions there are some helper functions providing such an approach. See [interactions.analysis](#).

2.4.2 Register Functions

The functions which are computing the attention levels have to be registered in order to ensure that they are called for the computation. This is done by using config entries. It is important to use the following keys:

- ANALYSIS_CALLBACK: name of the function which returns the attention level of a history after watching
- LIVE_ANALYSIS_CALLBACK: name of the function which returns the attention level of a user while watching

The following fixture shows an example of such config entries:

```

1  [
2      {
3          "pk": 2405,
4          "model": "core.configentry",
5          "fields": {
6              "interaction_method": 4,
7              "key": "ANALYSIS_CALLBACK",
8              "value": "interactions.ask_teacher.analysis.get_attention_level_per_history"
9          }
10     },
11     {
12         "pk": 2406,
13         "model": "core.configentry",
14         "fields": {
15             "interaction_method": 4,
16             "key": "LIVE_ANALYSIS_CALLBACK",
17             "value": "interactions.ask_teacher.analysis.get_attention_level"
18         }
19     },
20 ]

```

2.5 Interaction Framework

This is a basic introduction to the interaction framework. It shows how to develop an interaction method for LIVE. LIVE is based on django and because of that an interaction method is a django app.

For further details read the django documentation: <https://docs.djangoproject.com/en/1.11/>

Furthermore it is highly recommended to look into the code of existing interaction methods for a deeper understanding

2.5.1 General Steps

At first a django app has to be created inside the interaction app:

```
1 cd interactions
2 django-admin.py startapp <name of your interaction method>
```

After that add the appname to the ignore list in messages.sh. Furthermore it required to create a django app configuration and add it to the INSTALLED_APPS in LIVE/settings.py. Now create a url configuration for the app and call it in interactions/urls.py:

```
1 from django.conf.urls import url, include
2 from interactions.appname import urls as uappname
3
4 urlpatterns = [
5     ...
6     url(r'^appname/', include(uappname)),
7 ]
```

Finally it is required to register the app as an interaction method. For that a new entry in interactions/fixtures/settings.json has to be created:

```
1 [
2     {
3         "pk": 1, // running unique number
4         "model": "interactions.interactionmethod",
5         "fields": {
6             "title_de": "Titel der Interaktionsform ",
7             "app_name": "interactions.appname",
8             "title": "Title of the interaction method",
9             "pause_on": false, // true if the video should be paused when the interaction
10            ↪ is shown
11             "title_en": "Title of the interaction method",
12             "details_de": "Beschreibung der Interaktionsform",
13             "details": "Description of the interaction method",
14             "details_en": "Description of the interaction method",
15             "interactiontype": 1
16         }
17     },
18 ]
```

The interactiontype is the id of the type to use for this interaction method. There are four types:

1. Automatic: Interaction methods of this type are called automatically and in a random way a given number of times per hour.
2. Student Triggered: At this interaction type the student has to start the interaction by himself.
3. Teacher Triggered: While a live broadcasting is running the teacher could start interactions of this type.

4. Planned: The interactions of this type are shown at planned positions in the video.

2.5.2 Models

To save data there are the models. It is important to understand their structure because models are the key instrument for managing the interactions.

Each video is an `core.models.Event` and there the metadata and the concrete information of the video (streaming links) or of the live broadcasting is stored. If a user joins an Event a `core.models.JoinedUser` is created. Furthermore for each watched interval a `core.models.History` is used to log the joined timespans.

`interactions.models.Interaction` is the base for all interactions. It saves the name of the related view and the start position in the video. To display an interaction to a single student a `interactions.models.StudentInteraction` is used. It is linked to a `core.models.JoinedUser`. The `interactions.models.TeacherInteraction` shows an interaction to the teacher at a live broadcasting. For that it links to a user which has to be the teacher of the Event. The same is true for `interactions.models.ErrorToTeacher`.

To create interactions for students at pre-planned positions in the video there is `interactions.models.PlannedStudentInteraction`. It is a placeholder only and if a user joins an Event it is translated to a real `interactions.models.StudentInteraction`. This real student interaction is added to a list of the placeholder.

Every time an interaction is shown to a student or a teacher a `interactions.models.CallHistory` is created. It represents the state of an interaction.

For the models of an interaction method it is required to link to the interactions because this is the only way to load them in the standardized views shown below. Each callback view of an interaction is able to load `interactions.models.CallHistory` and `interactions.models.StudentInteraction` or `interactions.models.TeacherInteraction`.

Depending on the purpose of the model it should be linked to the correct interactions. In the case of a planned interaction and if the model holds data for all joined students it should link to a `interactions.models.PlannedStudentInteraction` because from there it is possible to query across all the related `interactions.models.StudentInteraction`. If it is no planned interaction but also holding data for all joined students it should link directly to all the `interactions.models.StudentInteraction`. These scenarios are typically for questions or something similar.

In comparison to that to store individual answers of the students a model should link to a `interactions.models.StudentInteraction` and a `interactions.models.CallHistory` because this combination is unique.

2.5.3 Views and Templates

Student Interaction

If an interaction is displayed to a student the related callback view is called with a HTTP-GET request. Any answer of the student has to be a HTTP-POST request. So a view should look like the following example:

```

1 from django.shortcuts import render
2 from django.contrib.auth.decorators import login_required
3 from django.http import HttpResponseRedirect, Http404
4 from django.core.urlresolvers import reverse
5 from django.utils import timezone
6
7 from interactions.student import StudentInteractionLoader
8 from interactions.utils import get_at_relative
9
10 @login_required
11 def index(request, student_i_id, call_id):
12     n = timezone.now()

```

(continues on next page)

(continued from previous page)

```

13     d = get_at_relative(post=request.POST, get=request.GET)
14     l = StudentInteractionLoader(user=request.user)
15     ok = l.load(student_i_id=student_i_id, call_id=call_id,
16                at_absolute=n, at_relative=d)
17     if not ok:
18         raise Http404
19
20     if request.method == 'POST':
21
22         # Here is the place to validate a form or react to the students
23         # response.
24
25         l.respond()
26         return HttpResponseRedirect(reverse('interactions.views.no_interactions'))
27
28     l.show()
29     params = {
30         'cid': int(call_id),
31         'siid': int(student_i_id)
32     }
33     return render(request, 'interactions/appname/index.html', params)

```

A template should look like this:

```

1  {% extends "interactions/interaction.html" %}
2  {% block script_dom_ready %}
3      registerPostSubmit("#answerForm");
4  {% endblock %}
5
6  {% block content %}
7      <form id="answerForm" method="post"
8          action="{% url 'interactions.appname.views.index' student_i_id=siid call_
9  ↪ id=cid %}">
10         {% csrf_token %}
11         ...
12     </form>
13     {% endblock %}

```

If the type of the interaction method is automatic it is required to provide the following settings. The first one indicates how often the interaction should be displayed per hour. The second is the name of the view.

```

1  [
2      {
3          "pk": 1101,
4          "model": "core.configentry",
5          "fields": {
6              "interaction_method": 1,
7              "value": "20",
8              "key": "TRIGGERS_PER_HOUR"
9          }
10     },
11     {
12         "pk": 1102,
13         "model": "core.configentry",
14         "fields": {
15             "interaction_method": 1,

```

(continues on next page)

(continued from previous page)

```

16     "value": "interactions.appname.views.index",
17     "key": "STUDENT_CALLBACK_VIEW"
18 }
19 },
20 ]

```

Teacher Interaction

Displaying an interaction to a teacher during a live broadcasting works similar to a student interaction. Again the callback view of the interaction is called:

```

1  from django.shortcuts import render
2  from django.contrib.auth.decorators import login_required
3  from django.http import HttpResponseRedirect, Http404
4  from django.core.urlresolvers import reverse
5  from django.utils import timezone
6  from interactions.teacher import TeacherInteractionLoader
7
8  @login_required
9  def index(request, teacher_i_id, call_id):
10     n = timezone.now()
11     l = TeacherInteractionLoader(user=request.user)
12     ok = l.load(teacher_i_id=teacher_i_id, call_id=call_id,
13               at_absolute=n)
14     if not ok:
15         raise Http404
16
17     if request.method == 'POST':
18
19         # Here is the place to validate a form or react to the teachers
20         # response.
21
22         l.respond()
23         return HttpResponseRedirect(reverse('interactions.views.no_interactions'))
24
25     l.show()
26     params = {
27         'cid': int(call_id),
28         'tiid': int(teacher_i_id),
29     }
30     return render(request, 'interactions/appname/index.html', params)

```

Also the template looks similar:

```

1  {% extends "interactions/interaction.html" %}
2  {% block content %}
3      <form id="answerForm" method="post"
4          action="{% url 'interactions.appname.views.index' teacher_i_id=tiid call_
5      id=cid %}">
6          {% csrf_token %}
7          ...
8      </form>
9  {% endblock %}

```

Initial Settings View

For planned interactions there are initial settings. Typically this is the creation of some questions at specific positions in the video. For that there are some views.

The first view is a page which displays the video to select the position and open an other view with the position as a parameter to create the question. Because this is done with a HTTP-POST request it is required to add a hidden parameter named `init_link` with the value 1. Furthermore it should list the created questions.

The second view is the view to create the questions and its interactions. In addition this view could be used to edit the question.

The third view marks the created interactions as ready. This means that after this no changes are possible and the interactions are ready to be shown to the students.

```

1  import datetime
2
3  from django.http import HttpResponseRedirect, Http404
4  from django.core.urlresolvers import reverse
5  from django.utils.translation import ugettext as _
6  from django.utils import timezone
7  from django.contrib.auth.decorators import login_required, user_passes_test
8  from django.shortcuts import render, get_object_or_404
9  from core.utils import is_in_teacher_group, type_to_mode, get_selector
10 from django.contrib import messages
11
12 from core.models import Event, JoinedUser, UsedLiveIM, UsedOnDemandIM
13 from interactions.utils import get_at_relative, get_duration, is_too_thight
14 from interactions.models import InteractionMethod, \
15     PlanedStudentInteraction, TeacherInteraction
16
17
18 @login_required
19 @user_passes_test(is_in_teacher_group)
20 def plan_questions(request, event_id, event_type):
21     e = get_object_or_404(Event, id=event_id)
22     mode = type_to_mode(event_type=event_type, event=e)
23     if mode is None:
24         raise Http404
25     d = e.get_duration(mode)
26     if d is None:
27         d = datetime.timedelta(seconds=0)
28
29     # load the created questions here and add them to p
30
31     p = {
32         'event': e,
33         'mode': mode,
34         'event_type': str(event_type),
35         'duration': int(d.total_seconds()),
36     }
37     p.update(get_selector(
38         request=request,
39         # name of the second view to create a question
40         view='interactions.appname.views.add_question',
41         event=e,
42         mode=mode))
43     return render(request,
44         'interactions/appname/plan_questions.html', p)

```

(continues on next page)

(continued from previous page)

```

45
46
47 @login_required
48 @user_passes_test(is_in_teacher_group)
49 def add_question(request, event_id, event_type, question_id=None):
50     e = get_object_or_404(Event, id=event_id)
51     mode = type_to_mode(event_type=event_type, event=e)
52     if mode is None:
53         raise Http404
54     d = get_at_relative(post=request.POST, get=request.GET)
55     if question_id is not None:
56         q = # load the question to edit here
57         d = q.planned_interaction.scheduled_start
58     if d is None:
59         raise Http404
60     try:
61         e_duration = get_duration(post=request.POST,
62                                   get=request.GET) - datetime.timedelta(seconds=20)
63         e_start = datetime.timedelta(seconds=20)
64         if d > e_duration or d < e_start:
65             messages.info(request,
66                             _('The position is too close to the beginning or the end!
67 →'))
68             return HttpResponseRedirect(reverse(
69                                     'interactions.planned_text_questions.views.plan_questions',
70                                     args=(e.id, str(event_type))))
71         if is_too_tight(e, mode, d, question_id):
72             messages.info(request,
73                             _('The position is too close to an other question!'))
74             return HttpResponseRedirect(reverse(
75                                     'interactions.planned_text_questions.views.plan_questions',
76                                     args=(e.id, str(event_type))))
77     except:
78         pass
79     try:
80         link = request.POST.__getitem__('init_link')
81     except:
82         link = None
83     if request.method == 'POST' and link is None:
84         if question_id is None:
85             form = # the form to handle the question
86         else:
87             form = # the form to handle the question to edit
88         if form.is_valid():
89             im = get_object_or_404(InteractionMethod,
90                                   title_en='xxx'
91                                   )
92             nq = form.save(commit=False)
93             cv_t = 'interactions.appname.views.teacher_answers'
94             cv_p = 'interactions.planned_text_questions.views.student_question'
95             try:
96                 if not nq.planned_interaction:
97                     pass
98             except:
99                 psi = PlannedStudentInteraction(
100                     callback_view=cv_p,

```

(continues on next page)

(continued from previous page)

```

100         scheduled_start=d,
101         interaction_method=im,
102         event=e,
103         mode=mode)
104     psi.save()
105     nq.planned_interaction = psi
106     if mode == JoinedUser.LIVE and nq.teacherinteraction is None:
107         ti = TeacherInteraction(
108             callback_view=cv_t,
109             scheduled_start=d,
110             interaction_method=im,
111             event=e)
112         ti.save()
113         nq.teacherinteraction = ti
114     nq.save()
115
116     if question_id is None:
117         messages.info(request,
118                        _('The question was created successfully!'))
119         return HttpResponseRedirect(reverse(
120             'interactions.appname.views.plan_questions',
121             args=(e.id, str(event_type))))
122     else:
123         messages.info(request,
124                        _('The question was edited successfully!'))
125         return HttpResponseRedirect(reverse(
126             'interactions.appname.views.plan_questions',
127             args=(e.id, str(event_type))))
128
129     else:
130         if question_id is None:
131             form = # the form to handle the question
132         else:
133             form = # the form to handle the question to edit
134     p = {
135         'event': e,
136         'mode': mode,
137         'event_type': str(event_type),
138         'position': d,
139         'duration': int(d.total_seconds()),
140         'form': form,
141         'qid': question_id
142     }
143     if question_id:
144         p['question'] = q
145     return render(request,
146                  'interactions/appname/add_question.html', p)
147
148
149 @login_required
150 @user_passes_test(is_in_teacher_group)
151 def mark_ready(request, event_id, event_type):
152     e = get_object_or_404(Event, id=event_id)
153     mode = type_to_mode(event_type=event_type, event=e)
154     im = get_object_or_404(InteractionMethod,
155                           title_en='xxx'

```

(continues on next page)

(continued from previous page)

```

156         )
157
158     if mode is JoinedUser.LIVE:
159         ulim = UsedLiveIM.objects.get(event=e.liveevent,
160                                     interaction_method=im)
161
162         ulim.ready = True
163         ulim.save()
164     elif mode is JoinedUser.ON_DEMAND:
165         uoim = UsedOnDemandIM.objects.get(event=e.ondemandevent,
166                                     interaction_method=im)
167
168         uoim.ready = True
169         uoim.save()
170     else:
171         raise Http404
172     messages.info(request,
173                 _('The xxx are ready!'))
174     return HttpResponseRedirect(reverse(
175         'core.views.details_event',
176         args=(e.id, )))

```

The following template is for the first view. At first it shows the video selector and after that it displays the created questions and the link to the mark ready view.

```

1  {% extends "core/base.html" %}
2
3  {% load i18n %}
4  {% load staticfiles %}
5
6  {% block breadcrumb %}
7      {{ block.super }}
8      <li><a href="{{ event_url }}">{% trans "Events" %}</a></li>
9
10     <li><a href="{% url 'core.views.on_demand_events' cat_id=event.category.id %}">
11         {{ event.category.long_name }}</a></li>
12     <li><a href="{% url 'core.views.details_event' event_id=event.id %}">
13         {{ event.title }}</a></li>
14     <li class="active">{% trans "Plan xxx" %}</li>
15 {% endblock %}
16
17
18 {% block add_libs %}
19 {% endblock %}
20
21 {% block title %}{% trans "Plan xxx" %}{% endblock %}
22
23 {% block content %}
24 <div class="event-details">
25     <h3>{% trans "Plan xxx" %}</h3>
26     <p class="headline-subtitle">
27         {% trans "Plan xxx for the event" %} "{{event.title}}"
28     </p>
29
30     {% include selector %}
31
32     <hr>
33     {% if planed_questions %}

```

(continues on next page)

(continued from previous page)

```

34 <h4>{% trans "Planned xxx" %}</h4>
35
36 {% comment %}
37     list the created questions here
38 {% endcomment %}
39
40 <form id="readyForm" method="post"
41     action="{% url 'interactions.appname.views.mark_ready'
42             event_id=event.id event_type=event_type %}">
43     {% csrf_token %}
44     <p><input type="submit" id="ready" class="btn btn-default btn-sm btn-custom"
45         value="{% trans "Mark as ready" %}" /></p>
46 </form>
47 {% endif %}
48
49 <div class="arrow-back">
50     <a href="{% url 'core.views.details_event' event_id=event.id %}">
51         <span class="glyphicon glyphicon-chevron-left"></span>
52         {% trans "Back" %}
53     </a>
54 </div>
55 </div>
56 {% endblock %}

```

The template for the second view should display the form to create or to edit the questions.

```

1  {% extends "core/base.html" %}
2
3  {% load i18n %}
4  {% load staticfiles %}
5  {% load widget_tweaks %}
6
7  {% block breadcrumb %}
8      {{ block.super }}
9      <li><a href="{{ event_url }}">{% trans "Events" %}</a></li>
10
11      <li><a href="{% url 'core.views.on_demand_events' cat_id=event.category.id %}">
12          {{ event.category.long_name }}</a></li>
13      <li><a href="{% url 'core.views.details_event' event_id=event.id %}">
14          {{ event.title }}</a></li>
15      <li class="active">{% trans "Plan xxx" %}</li>
16  {% endblock %}
17
18
19  {% block add_libs %}
20  {% endblock %}
21
22  {% block title %}
23  {% if qid %}
24      {% trans "Edit a xxx" %}
25  {% else %}
26      {% trans "Add a xxx" %}
27  {% endif %}
28  {% endblock %}
29  {% block script_dom_ready %}
30

```

(continues on next page)

(continued from previous page)

```

31 {% endblock %}
32
33 {% block content %}
34 <div class="event-details">
35     {% if qid %}
36     <h3>{% trans "Edit a xxx" %}!</h3>
37     <p class="headline-subtitle">
38         {% trans "Edit a xxx for the event" %} "{{ event.title }}"
39     </p>
40     {% else %}
41     <h3>{% trans "Add a xxx" %}!</h3>
42     <p class="headline-subtitle">
43         {% trans "Add a xxx for the event" %} "{{ event.title }}"
44     </p>
45     {% endif %}
46
47     {% trans "Position" %}: {{ position }}
48
49     <div class="form-container">
50     {% if qid %}
51     <form enctype="multipart/form-data" method="post"
52         action="{% url 'interactions.appname.views.add_question'
53             event_id=event.id event_type=event_type question_id=qid %}">
54     {% else %}
55     <form enctype="multipart/form-data" method="post"
56         action="{% url 'interactions.appname.views.add_question'
57             event_id=event.id event_type=event_type %}">
58     {% endif %}
59         {% csrf_token %}
60         <input type="hidden" name="at_relative" value="{{ duration }}" />
61
62         {% comment %}
63             display the form here
64         {% endcomment %}
65
66         <input type="submit" id="btn_create"
67             class="btn btn-default btn-sm btn-custom"
68             value="{% trans "Save" %}" />
69     </form>
70 </div>
71
72     <div class="arrow-back">
73         <a href="{% url 'interactions.appname.views.plan_questions'
74             event_id=event.id event_type=event_type %}">
75             <span class="glyphicon glyphicon-chevron-left"></span>
76             {% trans "Back" %}
77         </a>
78     </div>
79 </div>
80
81 {% endblock %}
82

```

It is required to provide the name of the first view in the config:

```
[
```

(continues on next page)

(continued from previous page)

```

2     {
3         "pk": 4801,
4         "model": "core.configentry",
5         "fields": {
6             "interaction_method": 8,
7             "key": "INITIAL_SETTINGS_VIEW",
8             "value": "interactions.appname.views.plan_questions"
9         }
10    },
11 ]

```

Student Details

Students should be able to see their answers and results after watching the video. For that the following config points to a function which returns links to such detail sites:

```

1 [
2     {
3         "pk": 4803,
4         "model": "core.configentry",
5         "fields": {
6             "interaction_method": 8,
7             "key": "STUDENT_DETAILS",
8             "value": "interactions.appname.utils.get_student_detail_links"
9         }
10    },
11 ]

```

Such a function looks like the following. It takes a joined user object as a parameter and returns a list of tuples. The first element of each tuple is the link and the second is a text for the link.

```

1 from django.utils.translation import gettext as _
2 from django.core.urlresolvers import reverse
3
4
5 def get_student_detail_links(joined):
6     l = reverse('interactions.appname.views.student_details',
7                 args=(joined.id, ))
8     t = _('xxx')
9     return [(l, t), ]

```

A view to display details for a student looks like the following. It gets the ID of a joined user as a parameter.

```

1 from django.http import Http404
2 from django.core.urlresolvers import reverse
3 from django.contrib.auth.decorators import login_required
4 from django.shortcuts import render, get_object_or_404
5 from core.models import JoinedUser
6
7 @login_required
8 def student_details(request, join_id):
9     j = get_object_or_404(JoinedUser, id=join_id)
10
11     # load the data for the details here and add them to p
12

```

(continues on next page)

(continued from previous page)

```

13     p = {'event': j.event, 'joined': j, 'mode': j.mode}
14     if j.mode == JoinedUser.LIVE:
15         p['type'] = 'live'
16     elif j.mode == JoinedUser.ON_DEMAND:
17         p['type'] = 'ondemand'
18     else:
19         raise Http404
20     return render(request,
21                 'interactions/appname/student_details.html', p)

```

A template for this view has the following structure:

```

1     {% extends "core/base.html" %}
2
3     {% load i18n %}
4     {% load staticfiles %}
5
6     {% block add_libs %}
7     {% endblock %}
8
9     {% block breadcrumb %}
10         {{ block.super }}
11         <li><a href="{{ event_url }}">{% trans "Events" %}</a></li>
12
13         <li><a href="{% url 'core.views.on_demand_events' cat_id=event.category.id %}">
14             {{ event.category.long_name }}</a></li>
15         <li><a href="{% url 'core.views.details_event' event_id=event.id %}">
16             {{ event.title }}</a></li>
17         <li class="active">{% trans "xxx" %}</li>
18     {% endblock %}
19
20     {% block title %}{{ event.title }}:
21     {% trans "xxx" %}{% endblock %}
22
23     {% block content %}
24     <div class="event-details">
25         <h3>{% trans "xxx" %}</h3>
26         <p class="headline-subtitle">
27             {% trans "xxx for the event" %} "{{ event.title }}"
28         </p>
29
30         {% comment %}
31             Show the data for the details here.
32         {% endcomment %}
33
34         <div class="arrow-back">
35             <a href="{% url 'core.views.analysis_event'
36                 event_id=event.id event_type=type join_id=joined.id %}">
37                 <span class="glyphicon glyphicon-chevron-left"></span>
38                 {% trans "Back" %}
39             </a>
40         </div>
41     </div>
42
43     {% endblock %}

```

Teacher Details

Very similar to the details for a student there are also details for the teacher. Such a detail site should display the answers and results of all students. Again a config points to a function which returns links to the detail views.

```
1  [
2      {
3          "pk": 4804,
4          "model": "core.configentry",
5          "fields": {
6              "interaction_method": 8,
7              "key": "TEACHER_DETAILS",
8              "value": "interactions.appname.utils.get_teacher_detail_links"
9          }
10     },
11 ]
```

Such a function returns a list of tuples and each tuple has at first the link and second a text for the link.

```
1  from django.utils.translation import ugettext as _
2  from django.core.urlresolvers import reverse
3
4
5  def get_teacher_detail_links(event, mode):
6      l = reverse('interactions.planned_text_questions.views.teacher_details',
7                  args=(event.id, mode))
8      t = _('xxx')
9      return [(l, t), ]
```

To show the details a view which looks like the following is used. Because it displays details for all students it takes the ID of an event and the mode as parameters.

```
1  from django.http import Http404
2  from django.core.urlresolvers import reverse
3  from django.contrib.auth.decorators import login_required, user_passes_test
4  from django.shortcuts import render, get_object_or_404
5  from core.utils import is_in_teacher_group,
6  from core.models import JoinedUser
7
8
9  @login_required
10 @user_passes_test(is_in_teacher_group)
11 def teacher_details(request, event_id, mode):
12     e = get_object_or_404(Event, id=event_id)
13
14     # load the data for the details and add them to p
15
16     p = {'event': e, 'mode': mode}
17     if mode == JoinedUser.LIVE:
18         p['type'] = 'live'
19     elif mode == JoinedUser.ON_DEMAND:
20         p['type'] = 'ondemand'
21     else:
22         raise Http404
23     return render(request,
24                   'interactions/appname/teacher_details.html', p)
```

For this view the following template should be used:

```

1  {% extends "core/base.html" %}
2
3  {% load i18n %}
4  {% load staticfiles %}
5
6  {% block add_libs %}
7  {% endblock %}
8
9  {% block breadcrumb %}
10     {{ block.super }}
11     <li><a href="{{ event_url }}">{% trans "Events" %}</a></li>
12
13     <li><a href="{% url 'core.views.on_demand_events' cat_id=event.category.id %}">
14         {{ event.category.long_name }}</a></li>
15     <li><a href="{% url 'core.views.details_event' event_id=event.id %}">
16         {{ event.title }}</a></li>
17     <li class="active">{% trans "xxx" %}</li>
18 {% endblock %}
19
20 {% block title %}{{ event.title }}:
21 {% trans "xxx" %}{% endblock %}
22
23 {% block content %}
24 <div class="event-details">
25     <h3>{% trans "xxx" %}</h3>
26     <p class="headline-subtitle">
27         {% trans "xxx for the event" %} "{{ event.title }}"
28     </p>
29
30     {% comment %}
31         display the data for the details here
32     {% endcomment %}
33
34     <div class="arrow-back">
35         <a href="{% url 'core.views.details_event' event_id=event.id %}">
36             <span class="glyphicon glyphicon-chevron-left"></span>
37             {% trans "Back" %}
38         </a>
39     </div>
40 </div>
41
42 {% endblock %}

```


PUBLICATIONS

- Wachtler, J. (2012). Echtzeit-Interaktionsformen fuer Web-Inhalte. Graz Universtiy of Technology.
- Ebner, M., Wachtler, J., & Holzinger, A. (2013). Introducing an information system for successful support of selective attention in online courses. In *Universal Access in Human-Computer Interaction. Applications and Services for Quality of Life* (pp. 153-162). Springer Berlin Heidelberg.
- Wachtler, J., & Ebner, M. (2014a, June). Support of Video-Based lectures with Interactions-Implementation of a first prototype. In *World Conference on Educational Multimedia, Hypermedia and Telecommunications* (Vol. 2014, No. 1, pp. 582-591).
- Wachtler, J., & Ebner, M. (2014b). Attention Profiling Algorithm for Video-Based Lectures. In *Learning and Collaboration Technologies. Designing and Developing Novel Learning Experiences* (pp. 358-367). Springer International Publishing.
- Wachtler, J., & Ebner, M. (2014c). Unterstützung von videobasiertem Unterricht durch Interaktionen-Implementierung eines ersten Prototyps. *Videos in der (Hochschul-) Lehre*, (3), 39.
- Wachtler, J., Geier, G., & Ebner, M. Design and Evaluation of a User Interface for an Interaction Supported Video Platform.
- Wachtler, J., & Ebner, M. (2015, June). Impacts of interactions in learning-videos: A subjective and objective analysis. In *EdMedia: World Conference on Educational Media and Technology* (Vol. 2015, No. 1, pp. 1611-1619).
- Wachtler, J., Khalil, M., Taraghi, B., & Ebner, M. (2016a). On Using Learning Analytics to Track the Activity of Interactive MOOC Videos. In *Proceedings of the workshop on Smart Environments and Analytics in Video-Based Learning (SE@ VBL), LAK2016*.
- Wachtler, J., Hubmann, M., Zöhrer, H., & Ebner, M. (2016b). An analysis of the use and effect of questions in interactive learning-videos. *Smart Learning Environments*, 3(1), 13.
- Wachtler, J., & Ebner, M. (2017, June). On Using Interactivity to Monitor the Attendance of Students at Learning-Videos. In *EdMedia: World Conference on Educational Media and Technology* (pp. 356-366). Association for the Advancement of Computing in Education (AACE).
- Wachtler, J., Scherz, M., & Ebner, M. (2019). Automatic authentication of students at an interactive learning-video platform. In *Proceedings of edmedia + innovate learning 2019* (pp. 715-728). Amsterdam, Netherlands: Association for the Advancement of Computing in Education (AACE).
- Wachtler, J. & Ebner, M. (2019). Scheduling interactions in learning videos: A state machine based algorithm. *International Journal of Learning Analytics and Artificial Intelligence for Education*, 1(1), 58-76.
- Wachtler, J. (2019). Interaction-based Support of Selective Attention in Online Courses. Graz Universtiy of Technology.

SOURCE CODE DOCUMENTATION

4.1 Manage-Files

The manage files are part of django and they are providing a command line interface to interact with the web app. The most important commands are those to manipulate the database and to run a development server.

For more information see the official documentation: <https://docs.djangoproject.com/en/1.11/ref/django-admin/>

4.1.1 manage.py

This is the command line interface to LIVE. It should be used for the development process. There are other interfaces for productive usage.

At first the settings module is loaded and the given command is executed.

4.1.2 manage_vagrant_lampy.py

This is the command line interface to LIVE. It is used for the simulation of the server lampy.tugraz.at in a vagrant box.

At first the settings module is loaded and the given command is executed.

4.1.3 manage_production_dev.py

This is the command line interface to LIVE. It is used for the development instance at the server lampy.tugraz.at

At first the settings module is loaded and the given command is executed.

4.1.4 manage_production.py

This is the command line interface to LIVE. It is used for the productive instance at the server lampy.tugraz.at

At first the settings module is loaded and the given command is executed.

4.2 Package: core

This is the main package of LIVE. It provides all basic functionalities of the web app. This includes the following things:

- user management
 - registration
 - authentication
 - password reset
 - edit user data
 - role management: normal users, teacher, researcher and admins
- configuration api
- admin interface to edit the configuration and to set user roles
- event management
 - create/edit/delete events: live broadcasting or on demand video
 - add interactions from the interaction module to the events
 - join users to events
- recording of joined and watched timespans for each user joined to an event
- after event polls
- news items and comments

4.2.1 Module: core.admin

This module registers some Core-Models at the admin site.

class `core.admin.CategoryAdmin(*args, **kwargs)`

Creating and editing categories of events.

class `core.admin.ConfigEntryAdmin(model, admin_site)`

Management of the config entries.

class `core.admin.EventAdmin(model, admin_site)`

Lists metadata of all events.

class `core.admin.LiveEventAdmin(model, admin_site)`

Shows the live events.

class `core.admin.LocalInfoInline(parent_model, admin_site)`

This class creates an inline admin interface for LocalInfo.

model

alias of `LocalInfo`

class `core.admin.NewsCommentInline(parent_model, admin_site)`

This admin inline handles the user comments.

model

alias of `NewsComment`

class `core.admin.NewsEntryAdmin(*args, **kwargs)`

Creating and editing news entries. The user comments are shown inline.

class `core.admin.OnDemandEventAdmin(model, admin_site)`

Lists all on demand events.

class `core.admin.UsedLiveIMInline(parent_model, admin_site)`

Lists all used live Interaction methods inline.

model

alias of *UsedLiveIM*

class `core.admin.UsedOnDemandIMInline(parent_model, admin_site)`

This admin inline lists all used on demand interaction methods.

model

alias of *UsedOnDemandIM*

class `core.admin.UserAdmin(model, admin_site)`

Extends the user admin interface with the LocalInfoInline.

class `core.admin.UserInfoInline(parent_model, admin_site)`

This class creates an inline admin interface for UserInfo.

model

alias of *UserInfo*

4.2.2 Module: `core.apps`

This module provides the main app configuration of django for the core package.

class `core.apps.CoreConfig(app_name, app_module)`

The configuration is stored in this class. It sets a name for the package.

4.2.3 Module: `core.config`

This module provides helper functions to access the configuration values stored in the database.

`core.config.get_config_bool(key, interaction_method=None)`

returns the config value identified by key and interaction_method as a boolean value

Keyword arguments:

- key – key of the requested config value
- interaction_method – matching interaction method

`core.config.get_config_bytes(key, interaction_method=None)`

returns the config value identified by key and interaction_method as bytes

Keyword arguments:

- key – key of the requested config value
- interaction_method – matching interaction method

`core.config.get_config_int(key, interaction_method=None)`

returns the config value identified by key and interaction_method as an int

Keyword arguments:

- key – key of the requested config value
- interaction_method – matching interaction method

`core.config.get_config_string(key, interaction_method=None)`

returns the config value identified by key and interaction_method as an unicode string

Keyword arguments:

- key – key of the requested config value
- interaction_method – matching interaction method

`core.config.get_configs(key)`

returns the config values identified by key.

Keyword arguments:

- key – key of the requested config value

4.2.4 Module: **core.context_processors**

This module provides some context processors.

`core.context_processors.analytics(request)`

Returns true or false indicating if the analytics code should be used from the active settings file.

Keyword arguments:

- request – the calling HttpRequest

`core.context_processors.debug_state(request)`

Returns the debug state.

Keyword arguments:

- request – the calling HttpRequest

`core.context_processors.messages_authenticated(request)`

Returns the messages to show at the login page.

Keyword arguments:

- request – the calling HttpRequest

`core.context_processors.messages_unauthenticated(request)`

Returns the messages to show at the login page.

Keyword arguments:

- request – the calling HttpRequest

`core.context_processors.version(request)`

Returns the current version of LIVE from the active settings file.

Keyword arguments:

- request – the calling HttpRequest

4.2.5 Module: **core.fields**

This module provides some model and form fields. They are adopted from Django 1.5.1.

class `core.fields.TimedeltaField(*args, **kwargs)`

Store Python's datetime.timedelta in an integer column.

formfield(`*args, **kwargs`)

Sets the form field.

from_db_value(*value, expression, connection, context=None*)

Converts the value from the database to a timedelta.

get_db_prep_lookup(*lookup_type, value, connection=None, prepared=False*)

This feature is not used.

get_db_prep_save(*value, connection=None, prepared=False*)

Converts a timedelta to an int.

get_internal_type()

Returns the internal type which is an IntegerField.

to_python(*value*)

Converts the value to a timedelta.

value_to_string(*obj*)

Returns an int string.

class `core.fields.TimedeltaFormField(*args, **kwargs)`

This is a form field for a timedelta.

clean(*value*)

Validates the form input.

value comes from `Timedelta.Widget.value_from_datadict()`: tuple of strings

class `core.fields.TimedeltaWidget(attrs=None)`

A widget to display a timedelta.

render(*name, value, attrs=None, renderer=None*)

Renders the widget to html.

value_from_datadict(*data, files, name*)

Get the value from a datadict. Don't throw `ValidationError` here, just return a tuple of strings.

class `core.fields.URLFormField(max_length=None, min_length=None, *args, **kwargs)`

This is a form field for the model field for URLs based on a CharField. It uses the same special validator to allow http and rtmp URLs.

to_python(*value*)

Returns the URL as a python string.

class `core.fields.URLModelField(verbose_name=None, name=None, **kwargs)`

This is a modal field to store an URL. It uses a special validator to allow http and rtmp URLs. It uses a CharField as the base class.

formfield(***kwargs*)

Defines a form field and returns it.

`core.fields.split_seconds(secs, inputs=['days', 'hours', 'minutes', 'seconds'], multiply=[86400, 3600, 60, 1], with_unit=False, remove_leading_zeros=False)`

Helper function to split the seconds of a timedelta.

4.2.6 Module: core.forms

This module provides all Django-Forms of the package core.

```
class core.forms.ContactForm(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)
```

This is a contact form to send a mail to the admin.

Fields:

- name – name of the mailer
- email – email of the mailer
- subject – subject of the mail
- message – message of the mail
- captcha – captcha to ensure human mailer

property media

Return all media required to render the widgets on this form.

```
class core.forms.EventForm(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, instance=None, use_required_attribute=None,
                           renderer=None)
```

This class represents a form for the creation or editing of a generic event. For that it uses the model Event.

Fields:

- title – The title of the event.
- description – the description of the event.
- details_url – an URL to a detail site of the event.
- category – select of the category.

property media

Return all media required to render the widgets on this form.

```
class core.forms.LiveEventForm(*args, **kwargs)
```

This class represents a form to create or edit an events live settings.

Fields:

- html5_webm_url – URL to a webm video for the html5 player
- html5_mp4_url – URL to a mp4 video for the html5 player
- html5_ogv_url – URL to a ogv video for the html5 player
- html5_hls_url – URL to a hls video for the html5 player
- scheduled_start – scheduled begin of the live event
- scheduled_end – scheduled end of the live event
- app_names – interaction methods for the event

clean()

Adds some further global validation. It checks if the scheduled_start date is in the future and if the scheduled_end date is after scheduled_start. Furthermore it checks if at least one automatic interaction method was chosen.

property media

Return all media required to render the widgets on this form.

```
class core.forms.NewsCommentForm(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)
```

This class represents a form to post a comment.

Fields:

- comment – the comment to post

property media

Return all media required to render the widgets on this form.

```
class core.forms.OnDemandEventForm(*args, **kwargs)
```

This class represents a form to create or edit an events ondemand settings.

Fields:

- youtube_id – youtube id of a video
- html5_webm_url – URL to a webm video for the html5 player
- html5_mp4_url – URL to a mp4 video for the html5 player
- html5_ogv_url – URL to a ogv video for the html5 player
- html5_hls_url – URL to a hls video for the html5 player
- app_names – interaction methods for the event

clean()

Adds some further global validation. It checks if at least one automatic interaction method was chosen.

property media

Return all media required to render the widgets on this form.

```
class core.forms.PEPAnswerForm(*args, **kwargs)
```

This class represents a form for the creation or editing of a PEPAnswer.

Fields:

- answer – given answer of a user to a poll

property media

Return all media required to render the widgets on this form.

```
class core.forms.PEPAnswerTextForm(*args, **kwargs)
```

This class represents a form for the creation or editing of a PEPAnswer.

Fields:

- answer – given answer of a user to a poll
- text – given text answer to a poll

property media

Return all media required to render the widgets on this form.

```
class core.forms.PEPPossibleAnswerForm(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, instance=None,
                                       use_required_attribute=None, renderer=None)
```

This class represents a form for the creation or editing of a PEPPossibleAnswer.

Fields:

- answer – possible answer (choice) of a poll

property media

Return all media required to render the widgets on this form.

```
class core.forms.PostEventPollForm(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, instance=None,  
                                use_required_attribute=None, renderer=None)
```

This class represents a form for the creation or editing of a PostEventPoll.

Fields:

- question – question of the poll
- image – image for the poll
- text – text of the poll
- show_after – show the poll after this watched percent

property media

Return all media required to render the widgets on this form.

```
class core.forms.UserCreateForm(*args, **kwargs)
```

This class represents a form to create a user.

In addition to the fields inherited from UserCreationForm (user name, password and password confirmation) it offers the following fields:

- first_name – the first name of the user
- last_name – the last name of the user
- email – the email address of the user

property media

Return all media required to render the widgets on this form.

```
class core.forms.UserCreateMultipleForm(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)
```

This form is used at the admin to create multiple users from a csv file.

Fields:

- csv – csv file to upload
- mail – true if the password should be mailed to the created users

property media

Return all media required to render the widgets on this form.

```
class core.forms.UserEditForm(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)
```

This class represents a form to edit user data.

Fields:

- first_name – the first name of the user
- last_name – the last name of the user

- email – the email address of the user
- timezone – the timezone of the user

property media

Return all media required to render the widgets on this form.

class `core.forms.Youtube2ClickForm(*args, **kwargs)`

This is a form for the Youtube2Click filter.

Fields:

- remember – remember selection

property media

Return all media required to render the widgets on this form.

4.2.7 Module: `core.middleware`

This module provides some middleware for the package core.

class `core.middleware.TimezoneMiddleware(get_response)`

This middleware class sets the timezone loaded from the database for the current user.

class `core.middleware.Youtube2ClickMiddleware(get_response)`

This middleware class redirects to a confirmation site if a youtube player should be shown.

4.2.8 Module: `core.models`

This module provides all Django-Database-Models of the package core.

class `core.models.Category(*args, **kwargs)`

This model represents a Category. It is used to group events.

Fields:

- short_name – short name of a category
- long_name – long name category
- text – description of a category

exception `DoesNotExist`

exception `MultipleObjectsReturned`

clean_fields(*exclude=None*)

Clean all fields and raise a `ValidationError` containing a dict of all validation errors if any occur.

get_deferred_fields()

Return a set containing names of deferred fields for this instance.

get_num_ondemand_events()

returns the number of ondemand events of a category

refresh_from_db(*using=None, fields=None*)

Reload field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The *using* parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

class `core.models.ConfigEntry(*args, **kwargs)`

This model represents a Config entry. It is used to store configuration values optionally linked to an interaction method.

Fields:

- `key` – key string of the config entry
- `value` – config value
- `interaction_method` – optional link to an interaction method

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `core.models.Event(*args, **kwargs)`

This model represents a generic event. It stores the metadata.

Fields:

- `title` – title of the event
- `description` – description of the event
- `details_url` – an URL to a details site of the event
- `teacher` – link to an user who is the teacher of the event
- `published_at` – date of publishing
- `active` – true if the event is active and shown
- `category` – link to the category

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_duration(*mode*)

returns the duration of the event given by mode.

Keyword arguments:

- `mode` – mode of the event

get_joined_users(*mode*)

returns the JoinedUsers

Keyword arguments:

- `mode` – mode of the event

is_joinable()

returns true if this event could be joined

is_live()

returns true if this is a live event

is_ondemand()

returns true if this is an on demand event

is_ready()

returns true if this event is ready to be shown public

is_user_joined(*user, mode*)

returns true if the given user is joined with the given mode.

Keyword arguments:

- *user* – user to check
- *mode* – mode to check

is_watchable()

returns true if this event could be watched

class `core.models.History(*args, **kwargs)`

This model represents a History of a JoinedUser. So it represents a joined timespan.

Fields:

- *join_relative* – position of joining in the video
- *join_absolute* – date of joining
- *leave_relative* – position of leaving in the video
- *leave_absolute* – date of leaving
- *joineduser* – link to the joined user
- *active* – true if the user is active in this history

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_absolute_length()

returns the absolute duration

get_join_percentage()

returns the join position relative to the event duration in percent.

get_last_percentage()

returns 100 - leave percentage

get_latest_student_interaction(*interaction_method, at_relative*)

returns last related student interaction of a given interaction method relative to a position in the video.

Keyword arguments:

- *interaction_method* – query only interactions of this method
- *at_relative* – position in the video

get_leave_percentage()

returns the leave position relative to the event duration in percent.

get_length()

returns relative duration

get_length_diff()

returns diff between relative and absolute duration

get_length_percentage()

returns the duration in percent

get_post_event_attention_level()

returns the attention level of the joined user during this history.

get_student_interactions(*interaction_method*)

returns related student interactions of a given interaction method.

Keyword arguments:

- *interaction_method* – return only interactions of this method

class `core.models.JoinedUser`(*args, **kwargs)

This model represents a JoinedUser. So it maps a user to a live event or to an ondemand event.

Constants:

- `LIVE` – indicates a mapping to a live event
- `ON_DEMAND` – indicates a mapping to an ondemand event
- `MODE_CHOICES` – possible choices for the mode: `LIVE` or `ON_DEMAND`

Fields:

- **mode** – indicates if the user is mapped to a live or an ondemand event by using `MODE_CHOICES`
- *last_lookup* – date of the last lookup for an interaction
- *expected_attention_level* – not used since v0.3
- *event* – a link to the generic event
- *student* – a link to a user who becomes a student

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_all_ims_for_mode()

returns all interaction methods of the related event.

get_attention_level(*at_relative*, *at_absolute*)

returns the current level of attention.

Keyword arguments:

- *at_relative* – current relative position in the video
- *at_absolute* – current absolute position in the video

get_avg_joined()

returns average joined timespan.

get_event_duration_1q_for_mode()

returns the duration of the first quarter.

get_event_duration_3q_for_mode()

returns the duration of the first three quarters.

get_event_duration_for_mode()

returns the duration.

get_event_for_mode()

returns the related concrete event object.

get_event_real_end()

returns *event_real_end*

get_event_real_start()

returns *event_real_start*

get_first_history(*relative=True*)

returns first history object.

Keyword arguments:

- **relative** – true (default) if relative time,
false if absolute time

get_ims_for_mode(*type_id*)

returns all interaction methods of the related event given by type_id.

Keyword arguments:

- **type_id** – id of the interaction type

get_last_history(*relative=True*)

returns last history object.

Keyword arguments:

- **relative** – true (default) if relative time,
false if absolute time

get_max_joined()

returns the longest joined timespan.

get_min_joined()

returns the shortest joined timespan.

get_num_history()

returns the number of related histories.

get_sum_joined()

returns the complete joined timespan.

get_watched_timespan()

returns the complete watched timespan.

get_watched_timespan_percent()

returns the complete watched timespan in percent.

is_here(*ref_timedelta*)

returns true if the joined user is currently active.

Keyword arguments:

- **ref_timedelta** – current position in the video

join(*at_relative*, *at_absolute*)

joins the user to the related event. this creates a new history object.

Keyword arguments:

- **at_relative** – current relative position in the video
- **at_absolute** – current absolute position in the video

leave(*at_relative*, *at_absolute*)

let the user leave the event.

Keyword arguments:

- **at_relative** – current relative position in the video
- **at_absolute** – current absolute position in the video

normalize_history()

normalizes the history by correcting impossible values caused by player errors.

class `core.models.LiveEvent(*args, **kwargs)`

This model represents a live event.

Constants: FLV, MP4, F4V, FORMAT_CHOICES, HTTP, RTMP, PROTOCOL_CHOICES

Fields:

- `html5_webm_url` – URL to a webm video for the html5 player
- `html5_mp4_url` – URL to a mp4 video for the html5 player
- `html5_ogv_url` – URL to a ogv video for the html5 player
- `html5_hls_url` – URL to a hls video for the html5 player
- `interaction_methods` – interaction methods for the event
- `event` – link to the generic event
- `scheduled_start` – scheduled start date of the live event
- `real_start` – real start date of the live event
- `scheduled_end` – scheduled end date of the live event
- `real_end` – real end date of the live event

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_attention_level(*at_relative*, *at_absolute*)

returns the current level of attention of all JoinedUsers

Keyword arguments:

- *at_relative* – current relative position in the video
- *at_absolute* – current absolute position in the video

get_duration()

returns the duration calculated from start and end date

get_joined_users()

returns the JoinedUsers

get_player_types()

returns the types of usable players: html, html5

get_teacher()

returns the teacher

get_title()

returns the title

is_ready()

returns true if this event is ready to be shown public

class `core.models.LocalInfo(*args, **kwargs)`

This model stores the timezone of a user.

Fields:

- `user` – one to one link to a user
- `timezone` – timezone code of the user

exception `DoesNotExist`

exception MultipleObjectsReturned

```
class core.models.NewsComment(*args, **kwargs)
```

This model represents a news comment.

Fields:

- comment – text of the comment
- posted_at – date of posting
- hidden – true if the comment should not be displayed
- news_entry – link to the news entry
- user – link to the posting user

exception DoesNotExist**exception MultipleObjectsReturned**

```
class core.models.NewsEntry(*args, **kwargs)
```

This model represents a news entry.

Fields:

- title – title of the news entry
- published_at – date of publishing the news entry
- active – true if the news entry is displayed
- short_text – short text of the news entry
- long_text – long text of the news entry
- is_message – will be shown as a message below the Navbar
- show_to_unauthenticated – message is shown at the login page
- show_to_students – message is shown to students
- show_to_teachers – message is shown to teachers
- show_to_researchers – message is shown to researchers
- show_to_admins – message is shown to admins

exception DoesNotExist**exception MultipleObjectsReturned**

```
clean_fields(exclude=None)
```

Clean all fields and raise a ValidationError containing a dict of all validation errors if any occur.

```
get_deferred_fields()
```

Return a set containing names of deferred fields for this instance.

```
refresh_from_db(using=None, fields=None)
```

Reload field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

visible_comments()

Returns the visible comments of the news entry.

class `core.models.OnDemandEvent(*args, **kwargs)`

This model represents an on demand event.

Constants: FLV, MP4, F4V, FORMAT_CHOICES, HTTP, RTMP, PROTOCOL_CHOICES

Fields:

- `youtube_id` – youtube id of a video
- `html5_webm_url` – URL to a webm video for the html5 player
- `html5_mp4_url` – URL to a mp4 video for the html5 player
- `html5_ogv_url` – URL to a ogv video for the html5 player
- `html5_hls_url` – URL to a hls video for the html5 player
- `interaction_methods` – interaction methods for the event
- `duration` – duration of the video
- `event` – link to the generic event

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_duration()

returns the duration.

get_joined_users()

returns the JoinedUsers

get_player_types()

returns the types of possible players as a list: youtube, html5

get_published_at()

returns date of publishing

get_teacher()

returns the teacher

get_title()

returns the title

is_ready()

returns true if this event is ready to be shown public

class `core.models.PEPAnswer(*args, **kwargs)`

This model represents the answer of a student to a poll.

Fields:

- `question` – link to a poll
- `answer` – link to the given answer choice
- `text` – optional text answer
- `answered_at` – date of answering
- `history` – related history in which the answer was given

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `core.models.PEPPossibleAnswer(*args, **kwargs)`

This model represents a possible choice for a poll.

Fields:

- `question` – link to a poll
- `answer` – possible choice for a poll

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `core.models.PostEventPoll(*args, **kwargs)`

This model represents a PostEventPoll.

Fields:

- `mode` – mode of the event
- `show_after` – show poll only after this watched percentage
- `event` – link to the generic event
- `published_at` – date of publishing the poll
- `active` – true if the poll is active to use
- `question` – poll question
- `image` – optional image for the question
- `text` – optional text for the poll, enables a text answer

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_event_for_mode()

returns the related concrete event object.

has_text()

returns true if there is a text

is_editable()

returns true if there are no answers

class `core.models.UsedLiveIM(*args, **kwargs)`

This model adds InteractionMethod to LiveEvent. This means that it defines the interaction methods of a live event.

Fields:

- `interaction_method` – the InteractionMethod to add
- `event` – the live event to add the InteractionMethod
- `ready` – true if the interaction method is ready to use

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_settings_link()

Returns a link to the initial settings view.

get_teacher_detail_links()

Returns a list of links to the teacher details views.

class `core.models.UsedOnDemandIM(*args, **kwargs)`

This model adds InteractionMethod to OnDemandEvent. This means that it defines the interaction methods of an ondemand event.

Fields:

- `interaction_method` – the InteractionMethod to add
- `event` – the ondemand event to add the InteractionMethod
- `ready` – true if the interaction method is ready to use

exception `DoesNotExist`

exception `MultipleObjectsReturned`

get_settings_link()

returns a link fo the initial settings view.

get_teacher_detail_links()

Returns a list of links to the teacher details views.

class `core.models.UserInfo(*args, **kwargs)`

This model stores additional information of a user.

Fields:

- `user` – one to one link to a user
- `organization` – organization of the user

exception `DoesNotExist`

exception `MultipleObjectsReturned`

4.2.9 Module: `core.research`

This module provides functions to create the data for the downloads of event related items.

`core.research.analyze_poll_questions(event, mode, show_headline=True)`

returns an analysis of all poll questions.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`core.research.category_list(show_headline=True)`

returns the short and the long name of all categories.

Keyword arguments:

- `show_headline` – true if the first item should be headlines

`core.research.list_poll_possible_answers(event, mode, show_headline=True)`

returns a list of all poll possible answers.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`core.research.list_poll_questions(event, mode, show_headline=True)`

returns a list of all poll questions.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`core.research.list_poll_student_answers(event, mode, show_headline=True, anonymous=False)`

returns a list of all poll student answers.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines
- `anonymous` – true if the username should be anonymous

`core.research.live_event_list(show_headline=True)`

returns all live events.

Keyword arguments:

- `show_headline` – true if the first item should be headlines

`core.research.ondemand_event_list(show_headline=True)`

returns all ondemand events.

Keyword arguments:

- `show_headline` – true if the first item should be headlines

`core.research.registration_statistic(show_headline=True)`

returns the numbers of registrations grouped by year and month.

Keyword arguments:

- `show_headline` – true if the first item should be headlines

`core.research.user_statistic()`

returns the number of users and the number of users with roles.

`core.research.users_per_second(event, mode, show_headline=True)`

returns a list of lists with the second, number of users, number of views and true/false for most and fewest watched. There is a sublist for each second of the video.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`core.research.users_per_second_statistics(data, show_headline=True)`

creates statistical data of the users_per_second.

Keyword arguments:

- `data` – the users_per_second
- `show_headline` – true if the first item should be headlines

`core.research.watch_statistic_per_user(event, mode, show_headline=True, anonymous=False)`

A list which contains lists with the username and the watched timespan in seconds and percent is returned.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines
- `anonymous` – true if username should be anonymous

`core.research.watch_statistic_per_user_statistics(data, show_headline=True)`

creates statistical data of the `watch_statistic_per_user`.

Keyword arguments:

- `data` – the `watch_statistic_per_user`
- `show_headline` – true if the first item should be headlines

`core.research.watched_timespans_per_user(event, mode, show_headline=True, anonymous=False)`

returns a list which contains a list for each joined history of each user. the sublist contains: username, join relative/absolute, leave relative/absolute, difference relative/absolute, attention level

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines
- `anonymous` – true if username should be anonymous

4.2.10 Package: `core.templatetags`

This package provides some additional template tags.

Module: `core.templatetags.group_tags`

Adds a tag to the template to check for user groups.

class `core.templatetags.group_tags.GroupCheckNode(group, nodelist_true, nodelist_false)`

Check to see if the currently logged in user belongs to a specific group. Requires the Django authentication contrib app and middleware.

Adopted from: <http://djangosnippets.org/snippets/2566/>

render(*context*)

Return the node rendered as a string.

`core.templatetags.group_tags.ifusergroup(parser, token)`

Check to see if the currently logged in user belongs to a specific group. Requires the Django authentication contrib app and middleware. Adopted from: <http://djangosnippets.org/snippets/2566/>

Usage: `{% ifusergroup Admins %} ... {% endifusergroup %}`, or

`{% ifusergroup Admins|Group1|"Group 2" %} ... {% endifusergroup %}`, or `{% ifusergroup Admins %} ... {% else %} ... {% endifusergroup %}`

Module: core.templatetags.urls

This module offers template tags to manipulate the URL.

`core.templatetags.urls.add_param(url, param, value, tab=None)`
 adds a parameter to an URL.

Keyword argument:

- url – URL to add a parameter
- param – parameter to add
- tab

4.2.11 Module: core.tests

core tests

```
class core.tests.EventTests(methodName='runTest')
    tests event model
    setUp()
        create test objects
    test_is_joinable()
        Tests is_joinable
    test_is_live()
        Tests is_live
    test_is_ondemand()
        Tests is_ondemand
    test_is_ondemand_and_is_live()
        Tests is_live and is_ondemand
    test_is_ready()
        Tests is_ready
    test_is_user_joined()
        Tests is_user_joined
class core.tests.JoinedUserTests(methodName='runTest')
    tests JoinedUser model
    setUp()
        create test objects
    test_event_real_end()
        Tests event_real_end
    test_event_real_start()
        Tests event_real_start
    test_is_here()
        Tests is_here
    test_model()
        Tests is_live
```

4.2.12 Module: `core.translation`

This module handles the model translation. It registers models which have content in multiple languages.

class `core.translation.CategoryTranslationOptions(model)`

Registers the fields of the Category model for translation.

class `core.translation.NewsEntryTranslationOptions(model)`

Registers the fields of the NewsEntry model for translation.

4.2.13 Module: `core.urls`

This module designs the urls of the package core.

4.2.14 Module: `core.utils`

This module provides some helper functions.

class `core.utils.PlayerDetector(request, concrete_event)`

detect_video player to use

detect()

performs the detection

get_error_return()

returns a HttpResponseRedirect in case of an error

get_params()

returns the player parameters

`core.utils.format_student_lookup(joined_user, ret_data, at_relative, computed, at_absolute)`

formats logging output of a student lookup

`core.utils.format_teacher_lookup(event, ret_data, at_relative, computed, at_absolute)`

formats logging output of a teacher lookup

`core.utils.get_by_py_path(py_path)`

Imports and returns a python callable.

Keyword arguments:

py_path – callable to load

`core.utils.get_interactions(type_ids)`

Returns a list of tuples of all InteractionMethods

Keyword arguments:

- type_ids – interaction types to use

`core.utils.get_selector(request, view, event, mode)`

Returns parameters in a dict for the selector used in the initial settings view.

Keyword arguments:

- request – calling HttpRequest
- view – the name of the view which is used to create a question
- event_mode – mode to convert
- event – related event

`core.utils.is_in_researcher_group(user)`

Checks if the given user is a researcher.

Keyword arguments:

user – a django User

`core.utils.is_in_teacher_group(user)`

Checks if the given user is a Teacher.

Keyword arguments:

user – a django User

`core.utils.lti_after_user_creation(user)`

adds timezone to user after lti creation

Keyword arguments:

- user – new user

`core.utils.mode_to_type(event_mode, event)`

converts mode used at models to event type used in URLs.

Keyword arguments:

- event_mode – mode to convert
- event – related event

`core.utils.paginator(request, elements, param_name)`

returns a paginator.

Keyword arguments:

- request – calling HttpRequest
- elements – list of elements to paginate
- param_name – name of the Http parameter with the current page number

`core.utils.redirect_get_params(view_name, **kwargs)`

Redirects to a view with get parameters

Keyword arguments:

- view_name – name of the view
- ****kwargs** – get parameters

`core.utils.type_to_mode(event_type, event)`

converts event type used in URLs to mode used at models.

Keyword arguments:

- event_type – type to convert
- event – related event

4.2.15 Module: core.validators

This module provides some validators. adopted from Django 1.5.1

class `core.validators.URLValidator(*args, **kwargs)`

Validates an URL with regex. It allows http, https, rtmp, rtmps

4.2.16 Package: core.views

This package encapsulates the views in different modules. All views are imported here to keep backward compatibility.

So all views are accessible with: `core.views.<view_name>`

Module: core.views.accounts

This module provides all account related views.

`core.views.accounts.bulk_user(request)`

Creates multiple users from a csv file. Used at admin.

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – test if user is a superuser

Template: `admin/bulk_user.html`

`core.views.accounts.edit_user(request)`

Handles user editing.

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in

Template: `registration/edit_user.html`

`core.views.accounts.extern_embed_login(request)`

This view logs an external user in. It is deprecated now. Use LTI.

Decorators:

- `csrf_exempt` – disable csrf protection

Keyword arguments:

- `request` – calling `HttpRequest`

`core.views.accounts.myacc(request)`

Displays the my account site.

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in

Template: `core/myacc.html`

`core.views.accounts.newuser(request)`

Handles user creation.

Keyword arguments:

- `request` – the calling `HttpRequest`

Template: `registration/create_user.html`

`core.views.accounts.password_change(request)`

Change Password.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in

`core.views.accounts.password_change_done(request)`

Password changed successfully.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in

Module: `core.views.basic`

This module offers some general views.

`core.views.basic.contact(request)`

Displays the contact formular.

Keyword arguments:

- request – the calling HttpRequest

Template: `core/contact.html`

`core.views.basic.index(request)`

Displays the main site. Shows new events and the news.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in

Template: `core/index.html`

`core.views.basic.news_details(request, news_id)`

Displays a news entry.

Keyword arguments:

- request – the calling HttpRequest
- news_id – news entry id

Decorators:

- login_required – the user has to be logged in

Template: `core/news_details.html`

`core.views.basic.player_set(request)`

sets the useable players.

Keyword arguments:

- request – the calling HttpRequest

`core.views.basic.playerdev(request)`

Displays the legal notices.

Keyword arguments:

- request – the calling HttpRequest

Template: core/playerdev.html

`core.views.basic.team(request)`

Displays the legal notices.

Keyword arguments:

- request – the calling HttpRequest

Template: core/team.html

`core.views.basic.youtube2click(request)`

Displays the confirmation dialog for youtube.

Keyword arguments:

- request – the calling HttpRequest

Template: core/youtube2click.html

Module: core.views.events

This module provides the views related to events.

`core.views.events.add_live_to_event(request, event_id)`

add live settings

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: core/add_live_to_event.html

`core.views.events.add_ondemand_to_event(request, event_id)`

add ondemand settings

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: core/add_ondemand_to_event.html

`core.views.events.add_to_event(request, event_id)`

select event type after create

Keyword arguments:

- request – the calling HttpRequest

- `event_id` – the id of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `core/add_to_event.html`

`core.views.events.analysis_event(request, event_id, event_type, join_id)`

displays the analysis of event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `join_id` – id of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `core/analysis_event.html`

`core.views.events.browse_teacher_errors(request, event_id, event_type)`

browse the teacher related errors.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the event
- `event_type` – type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `core/browse_teacher_errors.html`

`core.views.events.delete_event(request, event_id)`

Deletes an event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`core.views.events.details_event(request, event_id)`

Details of an event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event

Decorators:

- `login_required` – the user has to be logged in

Template: core/details_event.html

`core.views.events.edit_event_metadata(request, event_id=None)`

Handles the creation or editing of a generic event.

Keyword arguments:

- request – the calling HttpRequest
- event_id – id or None

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: core/edit_event.html

`core.views.events.manage_events(request)`

Displays a list of upcoming live events and all categories.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in

Template: core/manage_events.html

`core.views.events.my_created_events(request)`

Displays a list of my created Events.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: core/my_created_events.html

`core.views.events.my_joined_events(request)`

Displays a list of my joined Events.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in

Template: core/my_joined_events.html

`core.views.events.my_watched_events(request)`

Displays a list of my watched Events.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in

Template: core/my_watched_events.html

`core.views.events.on_demand_events(request, cat_id=None)`

Displays a list of ondemand events of a category

Keyword arguments:

- `request` – the calling `HttpRequest`
- `cat_id` – id of the category

Decorators:

- `login_required` – the user has to be logged in

Template: `core/on_demand_events.html`

`core.views.events.search_event_autocomplete(request)`

search for events

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in

`core.views.events.test_player(request, event_id, event_type, player)`

test for player type

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `player` – player to test

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`core.views.events.timeline_overall(request, event_id, event_type)`

Displays a timeline of the video and shows the number of joined and watching users.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `core/timeline_overall.html`

Module: `core.views.joined`

This module offers all views for the joined users.

`core.views.joined.is_started_event(request, event_id)`

This ajax-view return the status of the event: `not_started`, `started` or `stoped`

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event

Decorators:

- `login_required` – the user has to be logged in

`core.views.joined.join_event(request, event_id, event_type)`

Let the current user join an event to become a joined user.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – `live` or `ondemand`

Decorators:

- `login_required` – the user has to be logged in

`core.views.joined.leave_event(request, event_id, join_id)`

Let a user leave an event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `join_id` – the id of the `JoinedUser`

Decorators:

- `login_required` – the user has to be logged in

`core.views.joined.leave_handler(request, event_id, join_id)`

Handles action after leaving an event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `join_id` – the id of the `JoinedUser`

Decorators:

- `login_required` – the user has to be logged in

`core.views.joined.lookup_student(request, event_id, join_id)`

looks if there are student interacations to display.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `join_id` – the id of the `JoinedUser`

Decorators:

- login_required – the user has to be logged in

`core.views.joined.lookup_teacher(request, event_id)`

This ajax-view returns a live analysis as json. it looks if there are any teacher interactions to display.

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

`core.views.joined.pause_event(request, event_id, join_id)`

pause an event

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event
- join_id – the id of the JoinedUser

Decorators:

login_required – the user has to be logged in

`core.views.joined.resume_event(request, event_id, join_id)`

resume an event

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event
- join_id – the id of the JoinedUser

Decorators:

- login_required – the user has to be logged in

`core.views.joined.seek_event(request, event_id, join_id)`

seek in an event.

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event
- join_id – the id of the JoinedUser

Decorators:

- login_required – the user has to be logged in

`core.views.joined.show_to_student(request, event_id, join_id)`

shows the event to the student

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event
- join_id – the id of the JoinedUser

Decorators:

- `login_required` – the user has to be logged in

Template depends on player type: `core/show_student/<player>.html`

`core.views.joined.show_to_teacher(request, event_id)`

shows the live event to the teacher.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`core.views.joined.start_event(request, event_id)`

Starts a live event

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`core.views.joined.stop_event(request, event_id)`

Stops a live event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Module: `core.views.polls`

The views to handle the post event polls are provided by this module.

`core.views.polls.add_answer(request, event_id, event_type, poll_id, answer_id=None)`

Create/edit a possible choice for a poll.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `poll_id` – poll id to add the choice
- `answer_id` – id of the choice to edit, `None` to create

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `core/polls/add_answer.html`

`core.views.polls.add_poll(request, event_id, event_type, poll_id=None)`

Create/edit a new poll.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `poll_id` – poll id to edit, `None` to create

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `core/polls/add_poll.html`

`core.views.polls.analyze_poll(request, event_id, event_type, poll_id)`

Shows the analysis of a poll.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `poll_id` – poll id

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `core/polls/analyze_poll.html`

`core.views.polls.delete_answer(request, event_id, event_type, poll_id, answer_id)`

Deletes a possible choice of a poll.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `poll_id` – poll id
- `answer_id` – id of the choice to delete

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`core.views.polls.delete_poll(request, event_id, event_type, poll_id)`

Deletes a poll.

Keyword arguments:

- `request` – the calling `HttpRequest`

- `event_id` – the id of the event
- `event_type` – live or ondemand
- `poll_id` – poll id to delete

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`core.views.polls.manage_polls(request, event_id, event_type)`

Main page to manage the polls by the teacher. It lists all polls and offers possible actions.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `core/polls/manage_polls.html`

`core.views.polls.show_poll(request, event_id, event_type, poll_id, history_id)`

Displays a poll to a joined user after leaving.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `poll_id` – poll id
- `history_id` – related history of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `core/polls/show_poll.html`

`core.views.polls.switch_poll_state(request, event_id, event_type, poll_id)`

Switches the state (active or not active) of a poll.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – live or ondemand
- `poll_id` – poll id to switch

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

4.3 Package: LIVE

This package contains the settings for different usages and the corresponding wsgi files. Furthermore there is the main URL configuration.

4.3.1 Module: LIVE.settings

Django settings for LIVE.

To see the used settings read the code.

For more information on this file, see <https://docs.djangoproject.com/en/1.11/topics/settings/>

For the full list of settings and their values, see <https://docs.djangoproject.com/en/1.11/ref/settings/>

4.3.2 Module: LIVE.settings_production

Django settings for LIVE project (production version). It overwrites the main settings.

4.3.3 Module: LIVE.settings_production_dev

Django settings for LIVE project (production dev version). It overwrites the main settings.

4.3.4 Module: LIVE.settings_vagrant_lampy

Django settings for LIVE project (vagrant lampy). It overwrites the main settings.

4.3.5 Module: LIVE.urls

This is the main url configuration. It creates all auth related urls and includes the url files of the other packages.

4.4 Package: questionpool

4.4.1 Module: questionpool.admin

This module registers some models at the admin site for the questionpool package.

```
class questionpool.admin.PoolAdmin(model, admin_site)
```

Creating and editing pools.

```
class questionpool.admin.PoolMultipleChoiceQuestionInline(parent_model, admin_site)
```

This admin inline handles the pool multiple choice questions.

model

alias of *PoolMultipleChoiceQuestion*

```
class questionpool.admin.PoolProgrammingTaskInline(parent_model, admin_site)
```

This admin inline handles the pool programming tasks.

model

alias of *PoolProgrammingTask*

```
class questionpool.admin.PoolTextQuestionInline(parent_model, admin_site)
```

This admin inline handles the pool text questions.

model

alias of *PoolTextQuestion*

4.4.2 Module: questionpool.apps

This is the main django app configuration for the question pool.

```
class questionpool.apps.QuestionpoolConfig(app_name, app_module)
    sets some metadata
```

4.4.3 Module: questionpool.forms

This module holds the forms of the questionpool package.

```
class questionpool.forms.PoolMultipleChoiceQuestionForm(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,
    instance=None,
    use_required_attribute=None,
    renderer=None)
```

This class represents a form for the creation or editing of a pool multiple choice question. For that it uses the model *PoolMultipleChoiceQuestion*.

Fields:

- question – the actual question.
- image – an image for the question.
- show_answer – if the answer should be displayed to students.

property media

Return all media required to render the widgets on this form.

```
class questionpool.forms.PoolPossibleAnswerForm(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,
    instance=None, use_required_attribute=None,
    renderer=None)
```

This class represents a form for the creation or editing of the answer of a pool multiple choice question. For that it uses the model *PoolPossibleAnswer*.

Fields:

- answer – the actual answer.
- correct – if the answer is correct.

property media

Return all media required to render the widgets on this form.

```
class questionpool.forms.PoolProgrammingTaskForm(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,
    instance=None, use_required_attribute=None,
    renderer=None)
```

This class represents a form for the creation or editing of a pool programming task. For that it uses the model `PoolProgrammingTask`.

Fields:

- `task` – the actual task.
- `answer` – the answer to the task.
- `reference_output` – the reference output.
- `show_answer` – if the answer should be displayed to students.

property media

Return all media required to render the widgets on this form.

```
class questionpool.forms.PoolTextQuestionForm(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, instance=None,
                                              use_required_attribute=None, renderer=None)
```

This class represents a form for the creation or editing of a pool text question. For that it uses the model `PoolTextQuestion`.

Fields:

- `question` – the actual question.
- `question_image` – the image for a question.
- `answer` – the answer to the question.
- `answer_image` – the answer image for a question.
- `show_answer` – if the answer should be displayed to students.

property media

Return all media required to render the widgets on this form.

```
class questionpool.forms.QuestionpoolForm(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, instance=None,
                                          use_required_attribute=None, renderer=None)
```

This class represents a form for the creation or editing of a questionpool. For that it uses the model `Pool`.

Fields:

- `category` – The category of the pool.
- `name` – the name of the pool.
- `public` – public for all users or not.
- `description` – description of the pool.

property media

Return all media required to render the widgets on this form.

4.4.4 Module: questionpool.models

This module holds the models of the questionpool package.

class questionpool.models.Pool(*args, **kwargs)

This model represents a questionpool.

Fields:

- Category – a class representing various interactions
- name – text representing the name of the questionpool
- owner – link to the owner of the questionpool
- public – true if it is shown and can be used by other users
- description – text describing the questionpool

class Category(value)

An enumeration.

exception DoesNotExist

exception MultipleObjectsReturned

class questionpool.models.PoolMultipleChoiceQuestion(*args, **kwargs)

This model represents a mc question for the questionpool.

Fields:

- question – text of the question
- image – an optional image
- show_answer – true if the correct answer is shown to the students
- pool – link to the associated pool
- owner – link to the owner of the multiple choice question

exception DoesNotExist

exception MultipleObjectsReturned

class questionpool.models.PoolPossibleAnswer(*args, **kwargs)

This model represents a possible answer to a mc question.

Fields:

- question – link to the question via PlanedStudentInteraction
- answer – text of the possible answer
- correct – true if the possible answer is correct
- owner – link to the owner of the answer

exception DoesNotExist

exception MultipleObjectsReturned

class questionpool.models.PoolProgrammingTask(*args, **kwargs)

This model represents a programming task in the pool.

Fields:

- task – text representing the task
- answer – text representing the answer to the task

- `reference_output` – text representing the reference output
- `show_answer` – true if the correct answer is shown to the students
- `pool` – link to the associated pool
- `owner` – link to the owner of the programming task

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `questionpool.models.PoolTextQuestion(*args, **kwargs)`

This model represents a text question in the pool.

Fields:

- `question` – text representing the question
- `answer` – text representing the answer to the question
- `show_answer` – true if the correct answer is shown to the students
- `pool` – link to the associated pool
- `question_image` – image representing the text question
- `answer_image` – image representing the answer
- `owner` – link to the owner of the text question

exception `DoesNotExist`

exception `MultipleObjectsReturned`

4.4.5 Module: `questionpool.tests`

4.4.6 Module: `questionpool.urls`

This module defines the urls of the questionpool package.

4.4.7 Module: `questionpool.views`

This module offers the views of the questionpool package.

`questionpool.views.generate_pool_question(request)`

Displays a table for generating pools and questions

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is an admin

Template: `questionpool/generate_pool_question.html`

`questionpool.views.multiple_choice_question_answer_delete(request, pool_id=None, question_id=None, answer_id=None)`

Deletes a multiple choice question answer

Keyword arguments:

- request – the calling HttpRequest
- pool_id – id or None
- question_id – id or None
- answer_id – id or None

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

```
questionpool.views.multiple_choice_question_answer_edit_or_create(request,  
                                                                    question_id=None,  
                                                                    pool_id=None,  
                                                                    answer_id=None)
```

Handles the creation or editing of a multiple choice question answer.

Keyword arguments:

- request – the calling HttpRequest
- question_id – id or None
- pool_id – id or None
- answer_id – id or None

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: questionpool/multiple_choice_question_answer_edit_or_create.html

```
questionpool.views.multiple_choice_question_delete(request, pool_id=None, question_id=None)
```

Deletes a multiple choice question

Keyword arguments:

- request – the calling HttpRequest
- pool_id – id or None

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

```
questionpool.views.multiple_choice_question_edit_or_create(request, question_id=None,  
                                                            pool_id=None)
```

Handles the creation or editing of a multiple choice question.

Keyword arguments:

- request – the calling HttpRequest
- question_id – id or None
- pool_id – id or None

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: questionpool/multiple_choice_question_edit_or_create.html

`questionpool.views.pool_delete(request, pool_id=None)`

Deletes a pool

Keyword arguments:

- `request` – the calling `HttpRequest`
- `pool_id` – id or `None`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`questionpool.views.pool_detail(request, pool_id=None)`

Displays details of the pool and handles the creation of questions in the pool

Keyword arguments:

- `request` – the calling `HttpRequest`
- `pool_id` – id or `None`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `questionpool/pool_detail.html`

`questionpool.views.pool_edit_or_create(request, pool_id=None)`

Handles the creation or editing of a questionpool.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `pool_id` – id or `None`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `questionpool/pool_edit_or_create.html`

`questionpool.views.programming_task_delete(request, pool_id=None, task_id=None)`

Deletes a programming task

Keyword arguments:

- `request` – the calling `HttpRequest`
- `pool_id` – id or `None`
- `task_id` – id or `None`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`questionpool.views.programming_task_edit_or_create(request, task_id=None, pool_id=None)`

Handles the creation or editing of a programming task.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `task_id` – id or `None`

- `pool_id` – id or None

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `questionpool/task_edit_or_create.html`

`questionpool.views.select(request)`

Displays own and public pools

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `questionpool/select.html`

`questionpool.views.text_question_delete(request, pool_id: None, question_id=None)`

Deletes a text question

Keyword arguments:

- `request` – the calling `HttpRequest`
- `question_id` – id or None

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`questionpool.views.text_question_edit_or_create(request, question_id=None, pool_id=None)`

Handles the creation or editing of a text question.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `question_id` – id or None
- `pool_id` – id or None

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `questionpool/question_edit_or_create.html`

4.5 Package: interactions

This package provides the interaction framework. All required parts to implement an interaction method are located here.

Furthermore there is the framework to calculate the attention level.

4.5.1 Module: interactions.admin

This module registers some models at the admin site for the interactions package.

class interactions.admin.**InteractionMethodAdmin**(*args, **kwargs)

This class creates a translate-able admin interface for the interaction methods.

class interactions.admin.**InteractionTypeAdmin**(*args, **kwargs)

This class creates a translate-able admin interface for the interaction types.

4.5.2 Module: interactions.analysis

This module provides the analysis related helper functions. They are used by the interaction methods to create their level of attention.

interactions.analysis.**get_current_time_based_attention_level**(*joineduser, interaction_method, at_relative, at_absolute, success_until, failed_after, max_lookback*)

computes the current time based attention level of a joined user in the context of an interaction method.

Keyword arguments:

- *joineduser* – the joined user to analyze
- *interaction_method* – the interaction method as the context
- *at_relative* – current position
- *at_absolute* – current time
- *success_until* – timedelta up to which the attention level is 100
- *failed_after* – timedelta after which the attention level is 0
- *max_lookback* – max timedelta to look back in time

interactions.analysis.**get_post_event_time_based_attention_level**(*interaction_method, history, success_until, failed_after*)

computes the attention level of a history in the context of an interaction method.

Keyword arguments:

- *interaction_method* – the interaction method as the context
- *history* – history of a joined user
- *success_until* – timedelta up to which the attention level is 100
- *failed_after* – timedelta after which the attention level is 0

interactions.analysis.**time_based_attention_level**(*ref_timedelta, success_until, failed_after*)

computes a time based attention level.

Keyword arguments:

- *ref_timedelta* – timedelta to compute the attention level
- *success_until* – timedelta up to which the attention level is 100
- *failed_after* – timedelta after which the attention level is 0

4.5.3 Module: interactions.apps

This module offers the main django app configuration of the interactions package.

class `interactions.apps.InteractionsConfig`(*app_name*, *app_module*)

This class sets some metadata for the interactions package.

4.5.4 Module: interactions.forms

This module provides all Django-Forms of the interactions package.

class `interactions.forms.AnswerErrorForm`(*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*)

This class represents a form to answer an error.

Fields:

- `answer` – text of the answer

property `media`

Return all media required to render the widgets on this form.

4.5.5 Module: interactions.models

This module provides the models of the interactions package.

class `interactions.models.CallHistory`(*args, **kwargs)

This model represents a call of an interaction. It uses the states of the interactions. See `get_state()` to see the state machine.

Constants: `STATE_PENDING`, `STATE_WAITING`, `STATE_CLOSED`, `STATE_NOT_DONE`, `STATE_DONE`

Fields:

- `opened_at` – first creation of the call
- `closed_at` – closing date of the call
- `real_start_relative` – real start position of the call
- `real_start_absolute` – real start time of the call
- `response_relative` – response position of the call
- `response_absolute` – response time of the call
- `interaction` – link to the interaction of the call

exception `DoesNotExist`

exception `MultipleObjectsReturned`

close(*at_absolute*)

check state and close the call.

Keyword arguments:

- `at_absolute` – time of closing

get_absolute_end()

returns the absolute end of the call based on the state.

get_current_attention_level(*at_absolute, success_until, failed_after*)

returns the current time based attention level of the call.

Keyword arguments:

- *at_absolute* – current time
- *success_until* – timedelta up to which the attention level is 100
- *failed_after* – timedelta after which the attention level is 0

get_post_event_attention_level(*success_until, failed_after*)

returns the final time based attention level of the call.

Keyword arguments:

- *success_until* – timedelta up to which the attention level is 100
- *failed_after* – timedelta after which the attention level is 0

get_readable_state()

returns the state in a readable form.

get_response_delay()

returns the time between the start and the response of a call based on the state.

get_state()

returns the state.

not_done(*at_absolute*)

mark the call as not done.

Keyword arguments:

- *at_absolute* – time of closing

respond(*at_relative, at_absolute*)

react to the response to a call

Keyword arguments:

- *at_relative* – position of response
- *at_absolute* – time of response

show(*at_relative, at_absolute*)

mark the call as shown.

Keyword arguments:

- *at_relative* – position of showing
- *at_absolute* – time of showing

class `interactions.models.ErrorToTeacher`(*args, **kwargs)

This model represents an error to the teacher interaction. It inherits from the base interaction.

Fields:

- *subject* – the subject of the error
- *question* – the text of the error
- *answer* – the answer of the teacher to the error
- *event* – the event of the interaction
- *mode* – the mode of the event
- *joineduser* – the error reporting joined user

- **student_interaction** – the student interaction used to return the answer

exception DoesNotExist

exception MultipleObjectsReturned

class interactions.models.**Interaction**(*args, **kwargs)

This model represents an interaction. It is the base model for all types of interactions. Each interaction has a state based on the set fields of its call history.

Constants: **STATE_OPEN**, **STATE_PENDING**, **STATE_WAITING**, **STATE_CLOSED**, **STATE_NOT_DONE**, **STATE_DONE**

Fields:

- **callback_view** – the view of the interaction
- **scheduled_start** – the position in the video of the interaction
- **interaction_method** – link to the interaction method

exception DoesNotExist

exception MultipleObjectsReturned

close(*at_absolute*)

close the last call.

Keyword arguments:

- *at_absolute* – current time

get_last_call()

returns the last call

get_next_to_last_call()

returns the call before the last call

get_readable_state()

returns the state of the interaction in a readable form.

get_state()

returns the state of the interaction.

not_done(*at_absolute*)

mark the last call as not done.

Keyword arguments:

- *at_absolute* – current time

open_call(*at_absolute*)

open a new call. This creates a new call history.

Keyword arguments:

- *at_absolute* – current time

respond(*at_relative*, *at_absolute*)

react to the response to an interaction.

Keyword arguments:

- *at_relative* – current position
- *at_absolute* – current time

show(*at_relative*, *at_absolute*)

show the interaction by showing the call.

Keyword arguments:

- *at_relative* – current position
- *at_absolute* – current time

class `interactions.models.InteractionMethod(*args, **kwargs)`

This model represents the interaction methods.

Fields:

- **app_name** – module name of the interaction app in python-dotted-syntax
- **title** – the title of the app
- **details** – a description of the app
- **interactiontype** – an instance of `InteractionType`

exception `DoesNotExist`

exception `MultipleObjectsReturned`

clean_fields(*exclude=None*)

Clean all fields and raise a `ValidationError` containing a dict of all validation errors if any occur.

get_deferred_fields()

Return a set containing names of deferred fields for this instance.

refresh_from_db(*using=None*, *fields=None*)

Reload field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The *using* parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If *fields* is `None`, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

class `interactions.models.InteractionType(*args, **kwargs)`

This model represents the different types of interactions.

Fields:

- **name** – type name

exception `DoesNotExist`

exception `MultipleObjectsReturned`

clean_fields(*exclude=None*)

Clean all fields and raise a `ValidationError` containing a dict of all validation errors if any occur.

get_deferred_fields()

Return a set containing names of deferred fields for this instance.

refresh_from_db(*using=None*, *fields=None*)

Reload field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The *using* parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If *fields* is `None`, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

class `interactions.models.PlannedStudentInteraction(*args, **kwargs)`

This model represents a planned student interactions. It inherits from the base interaction and is a placeholder for planned interactions. On usage a real student interaction is created for each joined user.

Fields:

- `event` – the event of the interaction
- `mode` – the mode of the event
- `interactions` – the student interactions created on usage

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `interactions.models.StudentInteraction(*args, **kwargs)`

This model represents a student interaction. It inherits from the base interaction.

Fields:

- `joineduser` – the joined user of the interaction

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `interactions.models.TeacherInteraction(*args, **kwargs)`

This model represents a teacher interactions. It inherits from the base interaction.

Fields:

- `event` – the event of the interaction

exception `DoesNotExist`

exception `MultipleObjectsReturned`

4.5.6 Module: `interactions.student`

This module provides some student related interaction handlers.

class `interactions.student.StudentInteractionLoader(user)`

this class handles all student interaction related tasks.

clean()

sets all members to none. see the code for a list of members.

is_valid()

checks if the student interaction and its call is valid.

load(student_i_id, call_id, at_absolute, at_relative=None)

loads a student interaction and sets the members.

Keyword arguments:

- `student_i_id` – the id of the student interaction
- `call_id` – the id of the current call
- `at_relative` – current position
- `at_absolute` – current time

respond()

respond to the student interaction.

show()

shows the student interaction

`interactions.student.student_lookup(joined, at_relative, at_absolute)`

looks if there is a student interaction to display.

Keyword arguments:

- `joined` – joined user to look
- `at_relative` – current position
- `at_absolute` – current time

4.5.7 Module: `interactions.teacher`

This module provides some teacher related interaction handlers.

class `interactions.teacher.TeacherInteractionLoader(user)`

this class handles all teacher interaction related tasks.

clean()

sets all members to none. see the code for a list of members.

is_valid()

checks if the teacher interaction is valid.

load(`teacher_i_id, call_id, at_absolute, at_relative=None`)

loads the teacher interaction.

Keyword arguments:

- `teacher_i_id` – the id of the teacher interaction
- `call_id` – the id of the call
- `at_relative` – current position
- `at_absolute` – current time

respond()

respond to the teacher interaction.

show()

shows the teacher interaction

`interactions.teacher.error_lookup(event, at_relative, at_absolute)`

looks if there is a teacher error to display.

Keyword arguments:

- `event` – the event to lookup
- `at_relative` – current position
- `at_absolute` – current time

`interactions.teacher.teacher_lookup(event, at_relative, at_absolute)`

looks if there is a teacher interaction to display.

Keyword arguments:

- `event` – the event to lookup
- `at_relative` – current position

- `at_absolute` – current time

4.5.8 Module: `interactions.tests`

This modul provides some tests for the interaction handling.

class `interactions.tests.CallHistoryTest` (*methodName='runTest'*)

This testcase tests the call history.

test_state()

Tests the state of the call history.

class `interactions.tests.lookupTest` (*methodName='runTest'*)

This testcase tests the lookup for interactions.

setUp()

create test objects

test_state()

Tests the states of the interactions.

test_student_loader()

Tests the loading, showing and responding to student interactions.

test_student_lookup()

Tests the lookup for student interactions.

4.5.9 Module: `interactions.translation`

This module handles the model translation.

class `interactions.translation.InteractionMethodTranslationOptions` (*model*)

This class configures the translation of `InteractionMethod`.

class `interactions.translation.InteractionTypeTranslationOptions` (*model*)

This class configures the translation of `InteractionType`.

4.5.10 Module: `interactions.urls`

This module designs the urls of the interaction package. It also includes the urls of the interaction methods.

4.5.11 Module: `interactions.utils`

This module provides some helper functions.

`interactions.utils.get_at_relative` (*post, get*)

loads the relative time from post are get data.

Keyword arguments:

- *post* – dict of post parameters
- *get* – dict of get parameters

`interactions.utils.get_duration` (*post, get*)

loads the duration from post are get data.

Keyword arguments:

- *post* – dict of post parameters

- `get` – dict of get parameters

`interactions.utils.get_position_in_slot(slot, interval_l, positions, max_tries=5, spacer_time=45)`

This function computes a position for a random interaction.

Keyword arguments:

- `slot` – the number of the slot for the interaction
- `interval_l` – length of a slot
- `positions` – other interactions
- `max_tries` – maximum number of random searches
- `spacer_time` – seconds before and after interactions and slot borders

`interactions.utils.is_too_tight(event, mode, place, question_id)`

This function checks if there is enough space between interactions.

Keyword arguments:

- `event` – the event to test
- `mode` – the mode of the event
- `place` – the position of the question
- `question_id` – the id of the planned student interaction

4.5.12 Module: `interactions.views`

This module provides the Django-Views of the interactions package.

`interactions.views.answer_teacher_error(request, teacher_error_id)`

saves the answer to a teacher error during a live event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `teacher_error_id` – the id of the error to answer

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.views.answer_teacher_error_post_event(request, teacher_error_id)`

saves the answer to a teacher error.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `teacher_error_id` – the id of the error to answer

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/answer_teacher_error_post_event.html`

`interactions.views.build_plan(request, join_id)`

Builds a plan of automatic interactions for a `JoinedUser`. It also sets the duration of the stream and creates real student interactions from planned student interactions.

Keyword arguments:

- request – the calling HttpRequest
- join_id – JoinedUser id

Decorators:

- login_required – the user has to be logged in

`interactions.views.hello_interaction(request, student_i_id, call_id)`

Displays a wellcome student interaction.

Keyword arguments:

- request – the calling HttpRequest
- student_i_id – the id of the student interaction
- call_id – th id of the call

Decorators:

- login_required – the user has to be logged in

Template: `interactions/hello_interaction.html`

`interactions.views.hello_teacher_interaction(request, teacher_i_id, call_id)`

Displays a wellcome teacher interaction.

Keyword arguments:

- request – the calling HttpRequest
- teacher_i_id – the id of the teacher interaction
- call_id – th id of the call

Decorators:

- login_required – the user has to be logged in

Template: `interactions/hello_teacher_interaction.html`

`interactions.views.no_interactions(request)`

Displays a 'no interaction' dummy site. If this site is shown in the interaction dialog the dialog is hidden.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in

Template: `interactions/interaction.html`

`interactions.views.show_error_answer(request, student_i_id, call_id)`

Displays the answer to an error.

Keyword arguments:

- request – the calling HttpRequest
- student_i_id – the id of the student interaction
- call_id – th id of the call

Decorators:

- login_required – the user has to be logged in

Template: `interactions/error_answer.html`

4.5.13 Package: `interactions.ask_students`

This package holds an interaction method which offers the possibility to ask text based questions to the students at live broadcastings.

Module: `interactions.ask_students.analysis`

This module provides the analysis related classes.

`interactions.ask_students.analysis.get_attention_level(joineduser, at_relative, at_absolute)`

This function returns a timebased attention level for a joined user during an event.

Keyword arguments:

- `joineduser` – the joined user
- `at_relative` – current position
- `at_absolute` – current time

`interactions.ask_students.analysis.get_attention_level_per_history(history)`

This function returns the attention of a history of a joined user.

Keyword arguments:

- `history` – the history of a joined user

Module: `interactions.ask_students.apps`

This module holds the main django app configuration for this interaction method.

class `interactions.ask_students.apps.InteractionsAskStudentsConfig(app_name, app_module)`
sets some metadata

Module: `interactions.ask_students.forms`

This module provides the forms of this interaction method.

class `interactions.ask_students.forms.AnswerForm(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)`

This class represents a form answer a question.

Fields:

- `answer` – the answer to a question

property media

Return all media required to render the widgets on this form.

Module: interactions.ask_students.models

This module provides the models of this interaction method.

class interactions.ask_students.models.**AnswerFromStudents**(*args, **kwargs)

This model represents the answer of a student.

Fields:

- question – link to the related QuestionToStudents
- joined_user – link to the answering joined user
- answer – answer of the student
- answer_at – relative position of the answer in the video

exception DoesNotExist

exception MultipleObjectsReturned

class interactions.ask_students.models.**QuestionToStudents**(*args, **kwargs)

This model represents a question asked to students.

Fields:

- teacherinteraction – link to the related TeacherInteraction
- interactions – links to the related StudentInteractions
- question – the question to ask to the students
- answer – the answer of the teacher as a reference

exception DoesNotExist

exception MultipleObjectsReturned

Module: interactions.ask_students.tests

This module offers tests for this interaction method. Currently there are no tests.

Module: interactions.ask_students.urls

This module designs the urls of this interaction method.

Module: interactions.ask_students.utils

This module provides some helper functions for this interaction method.

interactions.ask_students.utils.**get_student_detail_links**(joined)

returns a list of URLs to detail sites for a student.

Keyword arguments:

- joined – the joined user to display the details

interactions.ask_students.utils.**get_teacher_detail_links**(event, mode)

returns an URL to a detail site for the teacher.

Keyword arguments:

- event – the event to show the details
- mode – the mode of the event

Module: interactions.ask_students.views

This module provides the Django-Views of the interaction method.

`interactions.ask_students.views.ask(request, event_id)`

This ajax view asks a question to students.

Keyword arguments:

- `request` – the calling HttpRequest
- `event_id` – event id

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.ask_students.views.student_answer(request, student_i_id, call_id)`

Displays the answer of the teacher to the students.

Keyword arguments:

- `request` – the calling HttpRequest
- `student_i_id` – id of the student interaction
- `call_id` – id of the current call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/ask_students/answer_student.html`

`interactions.ask_students.views.student_details(request, join_id)`

Displays the detail site for a student.

Keyword arguments:

- `request` – the calling HttpRequest
- `join_id` – id of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/ask_students/student_details.html`

`interactions.ask_students.views.student_question(request, student_i_id, call_id)`

Displays a question to the students with the answer form.

Keyword arguments:

- `request` – the calling HttpRequest
- `student_i_id` – id of the student interaction
- `call_id` – id of the current call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/ask_students/question_student.html`

`interactions.ask_students.views.teacher_answer(request, teacher_i_id, call_id)`

Displays a form to the teacher to enter the answer.

Keyword arguments:

- request – the calling HttpRequest
- teacher_i_id – id of the teacher interaction
- call_id – id of the current call

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: interactions/ask_students/answer_teacher.html

`interactions.ask_students.views.teacher_details(request, event_id, mode)`

Displays the detail site for the teacher.

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event
- mode – the mode of the event

Decorators:

- login_required – the user has to be logged in

Template: interactions/ask_students/details_teacher.html

`interactions.ask_students.views.teacher_question(request, teacher_i_id, call_id)`

Displays the answers of the students to the teacher.

Keyword arguments:

- request – the calling HttpRequest
- teacher_i_id – id of the teacher interaction
- call_id – id of the current call

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a teacher

Template: interactions/ask_students/question_teacher.html

4.5.14 Package: `interactions.ask_teacher`

This package provides an interaction method which enables a student to ask a question to the teacher.

Module: `interactions.ask_teacher.analysis`

This module provides the analysis related classes.

`interactions.ask_teacher.analysis.get_attention_level(joineduser, at_relative, at_absolute)`

This function returns a timebased attention level for a joined user during an event.

Keyword arguments:

- joineduser – the joined user
- at_relative – current position
- at_absolute – current time

`interactions.ask_teacher.analysis.get_attention_level_per_history(history)`

This function returns the attention of a history of a joined user.

Keyword arguments:

- `history` – the history of a joined user

Module: `interactions.ask_teacher.apps`

This is the main django app configuration for this interaction method.

class `interactions.ask_teacher.apps.InteractionsAskTeacherConfig(app_name, app_module)`
 sets some metadata

Module: `interactions.ask_teacher.forms`

This module provides the forms of this interaction method.

class `interactions.ask_teacher.forms.AnswerForm(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)`

This class represents a form to answer a question.

Fields:

- `answer` – the answer to a question

property media

Return all media required to render the widgets on this form.

Module: `interactions.ask_teacher.models`

This module provides the models of this interaction method.

class `interactions.ask_teacher.models.QuestionToTeacher(*args, **kwargs)`

This model represents a question to the teacher.

Fields:

- `asking_joined_user` – link to the question asking joined user
- `teacherinteraction` – link to the related TeacherInteraction
- `interactions` – related StudentInteractions
- `asked_relative` – relative asking position in the video
- `asked_absolute` – time of asking
- `question` – asked question
- `answer` – answer of the teacher

exception DoesNotExist

exception MultipleObjectsReturned

Module: interactions.ask_teacher.tests

This module offers tests for this interaction method. Currently there are no tests.

Module: interactions.ask_teacher.urls

This module designs the urls of this interaction method.

Module: interactions.ask_teacher.utils

This module provides the models of this interaction method.

`interactions.ask_teacher.utils.get_student_detail_links(joined)`

returns a list of URLs to detail sites for a student.

Keyword arguments:

- *joined* – the joined user to display the details

`interactions.ask_teacher.utils.get_teacher_detail_links(event, mode)`

returns an URL to a detail site for the teacher.

Keyword arguments:

- *event* – the event to show the details
- *mode* – the mode of the event

Module: interactions.ask_teacher.views

This module provides the Django-Views of this interaction method.

`interactions.ask_teacher.views.ask(request, join_id)`

This ajax view asks a question to the teacher.

Keyword arguments:

- *request* – the calling HttpRequest
- *join_id* – JoinedUser id

Decorators:

- *login_required* – the user has to be logged in

`interactions.ask_teacher.views.student_answer(request, student_i_id, call_id)`

Displays the answer of the teacher to the students.

Keyword arguments:

- *request* – the calling HttpRequest
- *student_i_id* – id of the student interaction
- *call_id* – id of the current call

Decorators:

- *login_required* – the user has to be logged in

Template: interactions/ask_teacher/answer_student.html

`interactions.ask_teacher.views.student_details(request, join_id)`

Displays the detail site for a student.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `join_id` – id of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/ask_teacher/student_details.html`

`interactions.ask_teacher.views.student_question(request, student_i_id, call_id)`

Displays a question to the teacher.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `student_i_id` – id of the student interaction
- `call_id` – id of the current call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/ask_teacher/question_student.html`

`interactions.ask_teacher.views.teacher_answer(request, question_id)`

Displays a form to answer a question after an event.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `question_id` – the id of the question to answer

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/ask_teacher/teacher_answer.html`

`interactions.ask_teacher.views.teacher_details(request, event_id, mode)`

Displays the detail site for the teacher.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `mode` – the mode of the event

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/ask_teacher/details_teacher.html`

`interactions.ask_teacher.views.teacher_question(request, teacher_i_id, call_id)`

Displays the question to the teacher and offers a form to answer it.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `teacher_i_id` – id of the teacher interaction
- `call_id` – id of the current call

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/ask_teacher/question_teacher.html`

4.5.15 Package: `interactions.attention`

This package contains an interaction method to enable a student to set their own level of attention by moving a slider.

Module: `interactions.attention.analysis`

This module provides the analysis related classes.

`interactions.attention.analysis.get_attention_level(joineduser, at_relative, at_absolute)`

Returns the attention level based on the last setting of the slider.

Keyword arguments:

- `joineduser` – the joined user
- `at_relative` – current position
- `at_absolute` – current time

`interactions.attention.analysis.get_attention_level_per_history(history)`

This function returns an attention level of a history by evaluating all set slider values during this history.

Keyword arguments:

- `history` – the history of a joined user

Module: `interactions.attention.apps`

This module provides the main django app configuration for this interaction method.

class `interactions.attention.apps.InteractionsAttentionConfig(app_name, app_module)`

Sets some metadata

Module: `interactions.attention.models`

This module provides the models of this interaction method.

class `interactions.attention.models.SetAttention(*args, **kwargs)`

This model represents an attention level set by the student.

Fields:

- `joined_user` – the joined user who sets his attention level
- `history` – related history
- `attention_level` – the set attention level
- `triggerd_at` – time of setting the attention level

exception `DoesNotExist`

exception `MultipleObjectsReturned`

Module: interactions.attention.tests

This module offers the tests for this interaction method. Currently there are no tests.

Module: interactions.attention.urls

This module designs the urls of this interaction method.

Module: interactions.attention.views

This module provides the Django-Views of the interaction method.

`interactions.attention.views.set_attention(request, join_id)`

This ajax view sets the attention level of a joined user.

Keyword arguments:

- `request` – the calling HttpRequest
- `join_id` – JoinedUser id

Decorators:

- `login_required` – the user has to be logged in

4.5.16 Package: interactions.planed_questions

This package represents an interaction method to show multiple choice questions at planned position in the video.

Module: interactions.planed_questions.admin

This module registers some models at the admin site. It is used to list the mc questions. Permission to add new questions is disabled.

class `interactions.planed_questions.admin.PlanedQuestionToStudentsAdmin(model, admin_site)`

Displays the mc questions at the admin site and shows the possible answers inline.

has_add_permission(*request, obj=None*)

Returns false to disable the adding of new answers.

class `interactions.planed_questions.admin.PossibleAnswerInline(parent_model, admin_site)`

Adds the possible answers to mc questions to the admin as an inline table.

Fields:

- `answer` – the possible answer
- `correct` – true if the answer is correct

has_add_permission(*request, obj*)

Returns false to disable the adding of new answers.

model

alias of `PossibleAnswer`

Module: `interactions.planed_questions.analysis`

This module provides the analysis related classes.

`interactions.planed_questions.analysis.get_attention_level(joineduser, at_relative, at_absolute)`

This function returns a timebased attention level for a joined user during an event.

Keyword arguments:

- `joineduser` – the joined user
- `at_relative` – current position
- `at_absolute` – current time

`interactions.planed_questions.analysis.get_attention_level_per_history(history)`

This function returns the attention of a history of a joined user.

Keyword arguments:

- `history` – the history of a joined user

Module: `interactions.planed_questions.apps`

This is the main django app configuration for this interaction method.

`class interactions.planed_questions.apps.InteractionsPlanedQuestionsConfig(app_name, app_module)`

sets some metadata

Module: `interactions.planed_questions.forms`

This module provides all Django-Forms of this interaction method.

`class interactions.planed_questions.forms.AnswerForm(*args, **kwargs)`

This class represents a form which is used to get the selected answers of the students.

Fields:

- `answer` – a checkbox list of all possible answers, saves the selected

property media

Return all media required to render the widgets on this form.

`class interactions.planed_questions.forms.PossibleAnswerForm(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, instance=None, use_required_attribute=None, renderer=None)`

This class represents a form for the creation or editing of a possible answer to a mc question.

Fields:

- `answer` – the text of the answer
- `correct` – true if the answer is correct

property media

Return all media required to render the widgets on this form.

```
class interactions.planed_questions.forms.QuestionForm(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, instance=None,
                                                    use_required_attribute=None,
                                                    renderer=None)
```

This class represents a form for the creation or editing of a mc question.

Fields:

- question – the text of the question
- image – an optional image
- show_answer – true if the correct answer is shown to the students

property media

Return all media required to render the widgets on this form.

Module: interactions.planed_questions.models

This module provides the models of this interaction method.

```
class interactions.planed_questions.models.ChoiceOfStudent(*args, **kwargs)
```

This model represents a selected answer of a student.

Fields:

- answer – link to the selected possible answer
- student_interaction – link to the related StudentInteraction
- call – link to the related call

exception DoesNotExist

exception MultipleObjectsReturned

```
class interactions.planed_questions.models.PlannedQuestionToStudents(*args, **kwargs)
```

This model represents a mc question asked to students.

Fields:

- teacherinteraction – link to the related TeacherInteraction
- planed_interaction – link to the related PlanedStudentInteraction
- question – text of the question
- image – an optional image
- show_answer – true if the correct answer is shown to the students

exception DoesNotExist

exception MultipleObjectsReturned

```
class interactions.planed_questions.models.PossibleAnswer(*args, **kwargs)
```

This model represents a possible answer to a mc question.

Fields:

- question – link to the question via PlanedStudentInteraction
- answer – text of the possible answer
- correct – true if the possible answer is correct

exception DoesNotExist

exception MultipleObjectsReturned

Module: `interactions.planed_questions.research`

This module offers functions to create the data for the downloads.

`interactions.planed_questions.research.analyze_planed_questions_student_answers(event, mode, show_headline=True)`

returns the number of correct/wrong answers for each planned mc question.

Keyword arguments:

- event – the event to analyze
- mode – the mode of the event
- show_headline – true if the first item should be headlines

`interactions.planed_questions.research.analyze_planed_questions_student_answers_statistics(data, show_headl`

returns some statistical data of the mc questions.

Keyword arguments:

- data – the watch_statistic_per_user
- show_headline – true if the first item should be headlines

`interactions.planed_questions.research.list_planed_questions(event, mode, show_headline=True)`

returns a list of all planned mc questions.

Keyword arguments:

- event – the event to analyze
- mode – the mode of the event
- show_headline – true if the first item should be headlines

`interactions.planed_questions.research.list_planed_questions_answers(event, mode, show_headline=True)`

returns a list of the answers to the mc questions.

Keyword arguments:

- event – the event to analyze
- mode – the mode of the event
- show_headline – true if the first item should be headlines

`interactions.planed_questions.research.list_planed_questions_student_answers(event, mode, show_headline=True, anonymous=False)`

returns a list of the given answers from the students to mc questions

Keyword arguments:

- event – the event to analyze
- mode – the mode of the event
- show_headline – true if the first item should be headlines
- anonymous – true if the username should be anonymous

Module: `interactions.planed_questions.tests`

This module holds the tests of this package. Currently there are no tests.

Module: `interactions.planed_questions.urls`

This module designs the urls of this interaction method.

Module: `interactions.planed_questions.utils`

This module provides some helper functions.

`interactions.planed_questions.utils.get_answers(question_id)`

Returns a list of tuples of all possible answers of a mc question.

Keyword arguments:

- question_id – id of the related mc question

`interactions.planed_questions.utils.get_num_correct_answers(question_id)`

Returns the number of correct possible answers.

Keyword arguments:

- question_id – id of the related mc question

`interactions.planed_questions.utils.get_student_detail_links(joined)`

returns a list of URLs to detail sites for a student.

Keyword arguments:

- joined – the joined user to display the details

`interactions.planed_questions.utils.get_teacher_detail_links(event, mode)`

returns an URL to a detail site for the teacher.

Keyword arguments:

- event – the event to show the details
- mode – the mode of the event

Module: `interactions.planed_questions.views`

This module provides the Django-Views of interaction method.

`interactions.planed_questions.views.add_answer(request, event_id, event_type, question_id, answer_id=None)`

Displays a form to create or to edit a possible answer to a mc question.

Keyword arguments:

- request – the calling HttpRequest
- event_id – id of the related event

- `event_type` – type of the event
- `question_id` – id of the mc question
- `answer_id` – id of the possible answer to edit, `None` for creation

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_questions/add_answer.html`

`interactions.planed_questions.views.add_question(request, event_id, event_type, question_id=None)`

Displays a form to create or to edit a mc question.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event
- `question_id` – id of the question to edit, `None` for creation

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_questions/add_question.html`

`interactions.planed_questions.views.delete_answer(request, event_id, event_type, question_id, answer_id)`

Deletes a possible answer to a mc question

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event
- `question_id` – id of the mc question
- `answer_id` – id of the possible answer to delete

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.planed_questions.views.delete_question(request, event_id, event_type, question_id)`

Deletes a mc question

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event
- `question_id` – id of the mc question to delete

Decorators:

- `login_required` – the user has to be logged in

- `user_passes_test` – check if the user is a teacher

`interactions.planed_questions.views.mark_ready(request, event_id, event_type)`

Marks the mc questions ready to be shown to the students.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.planed_questions.views.plan_questions(request, event_id, event_type)`

Displays the site to plan the mc questions.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_questions/plan_questions.html`

`interactions.planed_questions.views.student_list_my(request, join_id)`

Shows the analysis of the answers of a student.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `join_id` – id of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_questions/student_list_my.html`

`interactions.planed_questions.views.student_question(request, student_i_id, call_id)`

Displays a mc question to a student.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `student_i_id` – id of the student interaction
- `call_id` – id of the call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_questions/question_student.html`

`interactions.planed_questions.views.student_statistics(request, join_id)`

Displays a statistic of all answers.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `join_id` – id of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_questions/student_statistics.html`

`interactions.planed_questions.views.teacher_answers(request, teacher_i_id, call_id)`

Displays the answers of the students to the teacher at live broadcastings.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `teacher_i_id` – id of the teacher interaction
- `call_id` – id of the call

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_questions/teacher_answers.html`

`interactions.planed_questions.views.teacher_details(request, event_id, mode)`

Displays the results of the mc questions to the teacher.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `mode` – mode of the event

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_questions/teacher_details.html`

4.5.17 Package: `interactions.planed_text_questions`

This package represents an interaction method which shows text questions at planned positions.

Module: `interactions.planed_text_questions.admin`

This module registers some models at the admin site in a read-only way.

class `interactions.planed_text_questions.admin.PlannedTextQuestionToStudentsAdmin(model, admin_site)`

Shows the planned text questions at the admin site.

has_add_permission(*request, obj=None*)

Returns false to deactivate the creation.

Module: `interactions.planed_text_questions.analysis`

This module provides the analysis related classes.

`interactions.planed_text_questions.analysis.get_attention_level(joineduser, at_relative, at_absolute)`

This function returns a timebased attention level for a joined user during an event.

Keyword arguments:

- `joineduser` – the joined user
- `at_relative` – current position
- `at_absolute` – current time

`interactions.planed_text_questions.analysis.get_attention_level_per_history(history)`

This function returns the attention of a history of a joined user.

Keyword arguments:

- `history` – the history of a joined user

Module: `interactions.planed_text_questions.apps`

This is the main django app configuration for this interaction method.

`class interactions.planed_text_questions.apps.InteractionsPlannedTextQuestionsConfig(app_name, app_module)`

sets some metadata

Module: `interactions.planed_text_questions.forms`

This module provides all Django-Forms of this interaction method.

`class interactions.planed_text_questions.forms.AnswerForm(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)`

This class represents a form to enter the answer from the students.

Fields:

- `answer` – text of the student's answer

property media

Return all media required to render the widgets on this form.

`class interactions.planed_text_questions.forms.TextQuestionForm(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, instance=None, use_required_attribute=None, renderer=None)`

This class represents a form for the creation or editing of a text question.

Fields:

- question – text of the question
- question_image – optional image for the question
- answer – text of the reference answer
- answer_image – optional image for the answer
- show_answer – true if the reference answer is shown to the students

property media

Return all media required to render the widgets on this form.

Module: `interactions.planed_text_questions.models`

This module provides the models of this interaction method.

class `interactions.planed_text_questions.models.AnswerOfStudent(*args, **kwargs)`

This model represents the answer of a student.

Fields:

- question – link to the planed text question
- answer – text of the answer
- correct – true if the answer is correct
- student_interaction – link to the related student interaction
- call – link to the call

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `interactions.planed_text_questions.models.PlanedTextQuestionToStudents(*args, **kwargs)`

This model represents a text question asked to students.

Fields:

- teacherinteraction – link to the related TeacherInteraction
- planed_interaction – link to the PlanedStudentInteraction
- question – text of the question
- question_image – optional image for the question
- answer – text of the reference answer
- answer_image – optional image for the answer
- show_answer – true if the reference answer is shown to the students

exception `DoesNotExist`

exception `MultipleObjectsReturned`

Module: `interactions.planed_text_questions.research`

This module offers functions to create the data for the downloads.

`interactions.planed_text_questions.research.analyze_planed_text_questions`(*event, mode, show_headline=True*)

returns the number of correct/wrong/unchecked answers for each text question.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`interactions.planed_text_questions.research.analyze_planed_text_questions_statistics`(*data, show_headline=True*)

returns some statistical data of the text questions.

Keyword arguments:

- `data` – `analyze_planed_text_questions`
- `show_headline` – true if the first item should be headlines

`interactions.planed_text_questions.research.list_planed_text_questions`(*event, mode, show_headline=True*)

returns a list of all planned text questions.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`interactions.planed_text_questions.research.list_planed_text_questions_answers`(*event, mode, show_headline=True, anonymous=False*)

returns a list of the answers of the students to the text questions.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines
- `anonymous` – true if the username should be anonymous

Module: `interactions.planed_text_questions.tests`

This module holds the tests of this package. Currently there are no tests.

Module: interactions.planed_text_questions.urls

This module designs the urls of this interaction method.

Module: interactions.planed_text_questions.utils

This module provides some helper functions.

`interactions.planed_text_questions.utils.get_student_detail_links(joined)`
returns a list of URLs to detail sites for a student.

Keyword arguments:

- `joined` – the joined user to display the details

`interactions.planed_text_questions.utils.get_teacher_detail_links(event, mode)`
returns an URL to a detail site for the teacher.

Keyword arguments:

- `event` – the event to show the details
- `mode` – the mode of the event

Module: interactions.planed_text_questions.views

This module provides the Django-Views of this interaction method.

`interactions.planed_text_questions.views.add_question(request, event_id, event_type, question_id=None)`

Displays a form to create or to edit a text question.

Keyword arguments:

- `request` – the calling HttpRequest
- `event_id` – id of the related event
- `event_type` – type of the event
- `question_id` – id of the question to edit, None for creation

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_text_questions/add_question.html`

`interactions.planed_text_questions.views.delete_question(request, event_id, event_type, question_id)`

Deletes a text question

Keyword arguments:

- `request` – the calling HttpRequest
- `event_id` – id of the related event
- `event_type` – type of the event
- `question_id` – id of the text question to delete

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.planed_text_questions.views.mark_ready(request, event_id, event_type)`

Marks the text questions ready to be shown to the students.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.planed_text_questions.views.plan_questions(request, event_id, event_type)`

Displays the site to plan the text questions.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_text_questions/plan_questions.html`

`interactions.planed_text_questions.views.student_details(request, join_id)`

Shows the analysis of the answers of a student.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `join_id` – id of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_text_questions/student_details.html`

`interactions.planed_text_questions.views.student_question(request, student_i_id, call_id)`

Displays a text question to a student.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `student_i_id` – id of the student interaction
- `call_id` – id of the call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_text_questions/question_student.html`

`interactions.planed_text_questions.views.teacher_answers(request, teacher_i_id, call_id)`

Displays the answers of the students to the teacher at live broadcastings.

Keyword arguments:

- `request` – the calling `HttpRequest`

- `teacher_i_id` – id of the teacher interaction
- `call_id` – id of the call

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planned_text_questions/teacher_answers.html`

`interactions.planned_text_questions.views.teacher_details`(*request, event_id, mode, set_correct=None, set_wrong=None*)

Displays the results of the text questions to the teacher.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `mode` – mode of the event
- **`set_correct` – None or the id of an answer of a student to mark correct**
- **`set_wrong` – None or the id of an answer of a student to mark wrong**

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planned_text_questions/teacher_details.html`

4.5.18 Package: `interactions.planned_programming_tasks`

This package represents an interaction method which shows programming tasks at planned positions.

Module: `interactions.planned_programming_tasks.admin`

This module registers some models at the admin site in a read-only way.

class `interactions.planned_programming_tasks.admin.PlanedProgrammingTasksToStudentsAdmin`(*model, admin_site*)

Shows the planned programming tasks at the admin site.

has_add_permission(*request, obj=None*)

Returns false to deactivate the creation.

Module: `interactions.planned_programming_tasks.analysis`

This module provides the analysis related classes.

`interactions.planned_programming_tasks.analysis.get_attention_level`(*joineduser, at_relative, at_absolute*)

This function returns a timebased attention level for a joined user during an event.

Keyword arguments:

- `joineduser` – the joined user
- `at_relative` – current position

- `at_absolute` – current time

`interactions.planed_programming_tasks.analysis.get_attention_level_per_history(history)`

This function returns the attention of a history of a joined user.

Keyword arguments:

- `history` – the history of a joined user

Module: `interactions.planed_programming_tasks.apps`

This is the main django app configuration for this interaction method.

```
class interactions.planed_programming_tasks.apps.InteractionsPlannedProgrammingTasksConfig(app_name,
                                                                                             app_module)
    sets some metadata
```

Module: `interactions.planed_programming_tasks.forms`

This module provides all Django-Forms of this interaction method.

```
class interactions.planed_programming_tasks.forms.AnswerForm(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)
```

This class represents a form to enter the answer from the students.

Fields:

- `answer` – text of the student's answer

property media

Return all media required to render the widgets on this form.

```
class interactions.planed_programming_tasks.forms.ProgrammingTaskForm(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,
                                                                        instance=None,
                                                                        use_required_attribute=None,
                                                                        renderer=None)
```

This class represents a form for the creation or editing of a programming Task.

Fields:

- `question` – text of the programming task
- `answer` – text of the reference answer
- `show_answer` – true if the reference answer is shown to the students

- `reference_output` – text of the reference output

property media

Return all media required to render the widgets on this form.

```
class interactions.planed_programming_tasks.forms.TeacherNoteForm(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)
```

This class represents a form to enter the note to the student tasks.

Fields:

- `teacher_note` – text of the student’s answer

property media

Return all media required to render the widgets on this form.

Module: interactions.planed_programming_tasks.models

This module provides the models of this interaction method.

```
class interactions.planed_programming_tasks.models.PlanedProgrammingTasksToStudents(*args,  
                                                                                      **kwargs)
```

This model represents a programming task asked to students.

Fields:

- `teacherinteraction` – link to the related TeacherInteraction
- `planed_interaction` – link to the PlanedStudentInteraction
- `question` – text of the question
- `answer` – text of the reference answer
- `reference_output` – text of the reference output
- `show_answer` – true if the reference answer is shown to the students

exception DoesNotExist

exception MultipleObjectsReturned

```
class interactions.planed_programming_tasks.models.TaskAnswerOfStudent(*args, **kwargs)
```

This model represents the answer of a student.

Fields:

- `question` – link to the planed programming task
- `answer` – text of the answer
- `correct` – true if the answer is correct
- `student_interaction` – link to the related student interaction
- `call` – link to the call
- `output` – text output generated after execution of the code

- `teacher_note` – text of the note of the teacher

exception `DoesNotExist`

exception `MultipleObjectsReturned`

Module: `interactions.planed_programming_tasks.research`

This module offers functions to create the data for the downloads.

`interactions.planed_programming_tasks.research.analyze_planed_programming_tasks(event, mode, show_headline=True)`

returns the number of correct/wrong/unchecked answers for each programming task.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`interactions.planed_programming_tasks.research.analyze_planed_programming_tasks_statistics(data, show_headline=True)`

returns some statistical data of the text programming task.

Keyword arguments:

- `data` – `analyze_planed_text_questions`
- `show_headline` – true if the first item should be headlines

`interactions.planed_programming_tasks.research.list_planed_programming_tasks(event, mode, show_headline=True)`

returns a list of all planned programming tasks.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines

`interactions.planed_programming_tasks.research.list_planed_programming_tasks_answers(event, mode, show_headline=True, anonymous=False)`

returns a list of the answers of the students to the programming tasks.

Keyword arguments:

- `event` – the event to analyze
- `mode` – the mode of the event
- `show_headline` – true if the first item should be headlines
- `anonymous` – true if the username should be anonymous

Module: `interactions.planned_programming_tasks.tests`

This module holds the tests of this package. Currently, there are no tests.

Module: `interactions.planned_programming_tasks.urls`

This module designs the urls of this interaction method.

Module: `interactions.planned_programming_tasks.utils`

This module provides some helper functions.

`interactions.planned_programming_tasks.utils.get_student_detail_links(joined)`
returns a list of URLs to detail sites for a student.

Keyword arguments:

- `joined` – the joined user to display the details

`interactions.planned_programming_tasks.utils.get_teacher_detail_links(event, mode)`
returns an URL to a detail site for the teacher.

Keyword arguments:

- `event` – the event to show the details
- `mode` – the mode of the event

`interactions.planned_programming_tasks.utils.get_user_code_output(answer)`
returns the generated code output for an answer.

Keyword arguments:

- `answer` – the answer provided by an user.

Module: `interactions.planned_programming_tasks.views`

This module provides the Django-Views of this interaction method.

`interactions.planned_programming_tasks.views.add_task(request, event_id, event_type, question_id=None)`

Displays a form to create or to edit a programming task.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event
- `question_id` – id of the question to edit, `None` for creation

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planned_programming_task/add_task.html`

`interactions.planed_programming_tasks.views.delete_question(request, event_id, event_type, question_id)`

Deletes a programming task

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event
- `question_id` – id of the text question to delete

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.planed_programming_tasks.views.mark_ready(request, event_id, event_type)`

Marks the tasks ready to be shown to the students.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

`interactions.planed_programming_tasks.views.plan_tasks(request, event_id, event_type)`

Displays the site to plan the programming tasks.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `event_type` – type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_programming_tasks/plan_tasks.html`

`interactions.planed_programming_tasks.views.student_details(request, join_id)`

Shows the analysis of the answers of a student.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `join_id` – id of the joined user

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_programming_tasks/student_details.html`

`interactions.planed_programming_tasks.views.student_task(request, student_i_id, call_id)`

Displays a programming task to a student.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `student_i_id` – id of the student interaction
- `call_id` – id of the call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_programming_tasks/task_student.html`

`interactions.planed_programming_tasks.views.teacher_answers(request, teacher_i_id, call_id)`

Displays the answers of the students to the teacher at live broadcastings.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `teacher_i_id` – id of the teacher interaction
- `call_id` – id of the call

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a teacher

Template: `interactions/planed_programming_tasks/teacher_answers.html`

`interactions.planed_programming_tasks.views.teacher_details(request, event_id, mode,
set_correct=None,
set_wrong=None)`

Displays the results of the programming tasks to the teacher.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – id of the related event
- `mode` – mode of the event
- `set_correct` – **None or the id of an answer of a student to mark correct**
- `set_wrong` – **None or the id of an answer of a student to mark wrong**

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/planed_programming_tasks/teacher_details.html`

4.5.19 Package: interactions.report_problem

This package contains an interaction method which provides the possibility to report a problem to the teacher.

Module: interactions.report_problem.apps

This module holds the main django app configuration of this interaction method.

```
class interactions.report_problem.apps.InteractionsReportProblemConfig(app_name,  
                                                                    app_module)  
    sets some metadata
```

Module: interactions.report_problem.models

This module holds the models of this interaction method. Currently there are no models.

Module: interactions.report_problem.tests

This module offers the tests for this interaction method. Currently there are no tests.

Module: interactions.report_problem.urls

This module designs the urls of this interaction method.

Module: interactions.report_problem.views

This module provides the Django-Views of the interaction method.

```
interactions.report_problem.views.report(request, join_id)
```

This ajax view reports a problem to the teacher.

Keyword arguments:

- *request* – the calling HttpRequest
- *join_id* – JoinedUser id

Decorators:

- *login_required* – the user has to be logged in

4.5.20 Package: interactions.simple

This package encapsulates a simple interaction method. It automatically shows a random question.

Module: interactions.simple.analysis

This module provides the analysis related classes.

```
interactions.simple.analysis.get_attention_level(joineduser, at_relative, at_absolute)
```

This function returns a timebased attention level for a joined user during an event.

Keyword arguments:

- *joineduser* – the joined user
- *at_relative* – current position

- `at_absolute` – current time

`interactions.simple.analysis.get_attention_level_per_history(history)`

This function returns the attention of a history of a joined user.

Keyword arguments:

- `history` – the history of a joined user

Module: `interactions.simple.apps`

This is the main django app configuration for the simple interaction package.

`class interactions.simple.apps.InteractionsSimpleConfig(app_name, app_module)`

This class sets some metadata.

Module: `interactions.simple.models`

This module provides the models of this package. Currently there are no models.

Module: `interactions.simple.tests`

This module holds the tests of this package. Currently there are no tests.

Module: `interactions.simple.urls`

This module designs the urls of this package.

Module: `interactions.simple.views`

This module provides the Django-Views of the simple interaction package.

`interactions.simple.views.confirm(request, student_i_id, call_id)`

Handles question confirmation.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `student_i_id` – the id of the student interaction
- `call_id` – the id of the call

Decorators:

- `login_required` – the user has to be logged in

`interactions.simple.views.index(request, student_i_id, call_id)`

Displays a simple question to students.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `student_i_id` – the id of the student interaction
- `call_id` – the id of the call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/simple/ask.html`

4.5.21 Package: interactions.solve_captcha

This package provides an interaction method which shows a simple captcha.

Module: interactions.solve_captcha.analysis

This module provides the analysis related classes.

`interactions.solve_captcha.analysis.get_attention_level(joineduser, at_relative, at_absolute)`

This function returns a timebased attention level for a joined user during an event.

Keyword arguments:

- `joineduser` – the joined user
- `at_relative` – current position
- `at_absolute` – current time

`interactions.solve_captcha.analysis.get_attention_level_per_history(history)`

This function returns the attention of a history of a joined user.

Keyword arguments:

- `history` – the history of a joined user

Module: interactions.solve_captcha.apps

This is the main django app configuration of the solve_captcha interaction.

class `interactions.solve_captcha.apps.InteractionsSolveCaptchaConfig`(*app_name*,
app_module)

This class sets some metadata.

Module: interactions.solve_captcha.forms

This module provides the forms of this interaction method.

class `interactions.solve_captcha.forms.CaptchaForm`(*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)

This class represents a captcha form.

Fields:

- `captcha` – the captcha to solve

property media

Return all media required to render the widgets on this form.

Module: `interactions.solve_captcha.models`

This module provides the models of this interaction method. Currently there are no models.

Module: `interactions.solve_captcha.tests`

This module offers the tests of this interaction method. Currently there are no tests.

Module: `interactions.solve_captcha.urls`

This module designs the urls of this interaction method.

Module: `interactions.solve_captcha.views`

This module provides the Django-Views of the solve captcha interaction method.

`interactions.solve_captcha.views.index(request, student_i_id, call_id)`

Handles captcha solving.

Keyword arguments:

- `request` – the calling `HttpRequest`
- `student_i_id` – the id of the student interaction
- `call_id` – the id of the call

Decorators:

- `login_required` – the user has to be logged in

Template: `interactions/solve_captcha/captcha.html`

4.6 Package: research

This package provides the functionalities to enable researches to download all the data generated by LIVE as a spreadsheet.

4.6.1 Module: `research.admin`

This are the admin classes of the research module. Currently there are no admin classes.

4.6.2 Module: `research.apps`

This is the main django configuration of the research app.

class `research.apps.ResearchConfig(app_name, app_module)`

This config class sets some metadata.

4.6.3 Module: `research.models`

This module holds the models of the research package. Currently there are no models.

4.6.4 Module: `research.tests`

This module provides tests for the research package. Currently there are no tests.

4.6.5 Module: `research.urls`

This module defines the urls of the research package.

4.6.6 Module: `research.utils`

`research.utils.get_col_from_data_list(data, col)`

returns a column of a 2d matrix of lists.

Keyword arguments:

- `data` – the 2d matrix
- `col` – the number of the column to return

4.6.7 Module: `research.views`

This module offers the views of the research package.

`research.views.analyze_planned_programming_tasks_csv(request, event_id, event_type)`

download the analysis of the planned programming tasks as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.analyze_planned_programming_tasks_csv_statistics(request, event_id, event_type)`

download statistical data of the analysis of the programming tasks as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.analyze_planned_programming_tasks_show(request, event_id, event_type)`

shows the analysis of the programming tasks

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/analyze_planned_programming_tasks_show.html`

`research.views.analyze_planned_questions_student_answers_csv(request, event_id, event_type)`

download the analysis of the mc questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.analyze_planned_questions_student_answers_csv_statistics(request, event_id, event_type)`

downloads statistical data of the mc questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.analyze_planned_questions_student_answers_show(request, event_id, event_type)`

shows the analysis of the mc questions

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/analyze_planned_questions_student_answers_show.html`

`research.views.analyze_planed_text_questions_csv(request, event_id, event_type)`

download the analysis of the text questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.analyze_planed_text_questions_csv_statistics(request, event_id, event_type)`

download statistical data of the analysis of the text questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.analyze_planed_text_questions_show(request, event_id, event_type)`

shows the analysis of the text questions

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/analyze_planed_text_questions_show.html`

`research.views.analyze_poll_questions_csv(request, event_id, event_type)`

download the analysis of the polls as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.analyze_poll_questions_show(request, event_id, event_type)`

shows the analysis to the polls

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/analyze_poll_questions_show.html`

`research.views.category_list_csv(request)`

download the category list as csv

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.category_list_show(request)`

shows the category list

Keyword arguments:

- `request` – the calling `HttpRequest`

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/category_list_show.html`

`research.views.list_planned_programming_tasks_answers_csv(request, event_id, event_type)`

download the answers to programming tasks as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_planned_programming_tasks_answers_show(request, event_id, event_type)`

shows the answers to the programming tasks

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_planed_programming_tasks_answers_show.html`

`research.views.list_planed_programming_tasks_csv(request, event_id, event_type)`

download the planned programming tasks as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_planed_programming_tasks_show(request, event_id, event_type)`

displays the planned programming tasks

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_planed_text_questions_show.html`

`research.views.list_planed_questions_answers_csv(request, event_id, event_type)`

download the possible answers to mc questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_planed_questions_csv(request, event_id, event_type)`

download planned mc questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in

- `user_passes_test` – check if the user is a researcher

`research.views.list_planed_questions_show(request, event_id, event_type)`

shows the planned mc questions

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_planed_questions_show.html`

`research.views.list_planed_questions_student_answers_csv(request, event_id, event_type)`

download the given answers to mc questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_planed_questions_student_answers_show(request, event_id, event_type)`

shows the given answers to mc questions

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_planed_questions_student_answers_show.html`

`research.views.list_planed_text_questions_answers_csv(request, event_id, event_type)`

download the answers to text questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_planed_text_questions_answers_show(request, event_id, event_type)`

shows the answers to text questions

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_planed_text_questions_answers_show.html`

`research.views.list_planed_text_questions_csv(request, event_id, event_type)`

download the planned text questions as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_planed_text_questions_show(request, event_id, event_type)`

displays the planned text questions

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_planed_text_questions_show.html`

`research.views.list_poll_possible_answers_csv(request, event_id, event_type)`

download the questions of the polls as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_poll_questions_csv(request, event_id, event_type)`

download the questions of the polls as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_poll_questions_show(request, event_id, event_type)`

shows the polls

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_poll_questions_show.html`

`research.views.list_poll_student_answers_csv(request, event_id, event_type)`

download the questions of the polls as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.list_poll_student_answers_show(request, event_id, event_type)`

shows the given answers to the polls

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/list_poll_student_answers_show.html`

`research.views.live_event_list_csv(request)`

download the live events as csv

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

`research.views.live_event_list_show(request)`

shows the live events

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

Template: `research/live_event_list_show.html`

`research.views.ondemand_event_list_csv(request)`

download the ondemand events as csv

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

`research.views.ondemand_event_list_show(request)`

shows the ondemand events

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

Template: `research/ondemand_event_list_show.html`

`research.views.registration_csv(request)`

download the registration statistic as csv

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

`research.views.registration_show(request)`

displays the registration statistic

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

Template: research/registration_show.html

`research.views.select(request)`

Displays the main research site.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

Template: research/select.html

`research.views.user_csv(request)`

download user statistic as csv

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

`research.views.user_show(request)`

displays the user statistic.

Keyword arguments:

- request – the calling HttpRequest

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

Template: research/user_show.html

`research.views.users_per_second_csv(request, event_id, event_type)`

download the users per second as csv

Keyword arguments:

- request – the calling HttpRequest
- event_id – the id of the event
- event_type – the type of the event

Decorators:

- login_required – the user has to be logged in
- user_passes_test – check if the user is a researcher

`research.views.users_per_second_csv_statistics(request, event_id, event_type)`

download statistical data of the users per second as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.users_per_second_show(request, event_id, event_type)`

shows the users per second

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/users_per_second_show.html`

`research.views.watch_statistic_per_user_csv(request, event_id, event_type)`

download the watch statistic per user as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged

`research.views.watch_statistic_per_user_csv_statistics(request, event_id, event_type)`

download statistical data of the watch statistic per user as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in

`research.views.watch_statistic_per_user_show(request, event_id, event_type)`

displays the watch statistic per user

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event

- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/watch_statistic_per_user_show.html`

`research.views.watched_timespans_per_user_csv(request, event_id, event_type)`

download the watched timespans per user as csv

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

`research.views.watched_timespans_per_user_show(request, event_id, event_type)`

shows the watched timespans per user

Keyword arguments:

- `request` – the calling `HttpRequest`
- `event_id` – the id of the event
- `event_type` – the type of the event

Decorators:

- `login_required` – the user has to be logged in
- `user_passes_test` – check if the user is a researcher

Template: `research/watched_timespans_per_user_show.html`

PYTHON MODULE INDEX

C

- core, 26
- core.admin, 26
- core.apps, 27
- core.config, 27
- core.context_processors, 28
- core.fields, 28
- core.forms, 30
- core.middleware, 33
- core.models, 33
- core.research, 42
- core.templatetags, 44
- core.templatetags.group_tags, 44
- core.templatetags.urls, 45
- core.tests, 45
- core.translation, 46
- core.urls, 46
- core.utils, 46
- core.validators, 47
- core.views, 48
- core.views.accounts, 48
- core.views.basic, 49
- core.views.events, 50
- core.views.joined, 54
- core.views.polls, 56

I

- interactions, 66
- interactions.admin, 67
- interactions.analysis, 67
- interactions.apps, 68
- interactions.ask_students, 77
- interactions.ask_students.analysis, 77
- interactions.ask_students.apps, 77
- interactions.ask_students.forms, 77
- interactions.ask_students.models, 78
- interactions.ask_students.tests, 78
- interactions.ask_students.urls, 78
- interactions.ask_students.utils, 78
- interactions.ask_students.views, 79
- interactions.ask_teacher, 80
- interactions.ask_teacher.analysis, 80
- interactions.ask_teacher.apps, 81
- interactions.ask_teacher.forms, 81
- interactions.ask_teacher.models, 81
- interactions.ask_teacher.tests, 82
- interactions.ask_teacher.urls, 82
- interactions.ask_teacher.utils, 82
- interactions.ask_teacher.views, 82
- interactions.attention, 84
- interactions.attention.analysis, 84
- interactions.attention.apps, 84
- interactions.attention.models, 84
- interactions.attention.tests, 85
- interactions.attention.urls, 85
- interactions.attention.views, 85
- interactions.forms, 68
- interactions.models, 68
- interactions.planned_programming_tasks, 98
- interactions.planned_programming_tasks.admin, 98
- interactions.planned_programming_tasks.analysis, 98
- interactions.planned_programming_tasks.apps, 99
- interactions.planned_programming_tasks.forms, 99
- interactions.planned_programming_tasks.models, 100
- interactions.planned_programming_tasks.research, 101
- interactions.planned_programming_tasks.tests, 102
- interactions.planned_programming_tasks.urls, 102
- interactions.planned_programming_tasks.utils, 102
- interactions.planned_programming_tasks.views, 102
- interactions.planned_questions, 85
- interactions.planned_questions.admin, 85
- interactions.planned_questions.analysis, 86
- interactions.planned_questions.apps, 86
- interactions.planned_questions.forms, 86
- interactions.planned_questions.models, 87
- interactions.planned_questions.research, 88
- interactions.planned_questions.tests, 89
- interactions.planned_questions.urls, 89
- interactions.planned_questions.utils, 89
- interactions.planned_questions.views, 89
- interactions.planned_text_questions, 92
- interactions.planned_text_questions.admin,

92
interactions.planed_text_questions.analysis,
93
interactions.planed_text_questions.apps, 93
interactions.planed_text_questions.forms,
93
interactions.planed_text_questions.models,
94
interactions.planed_text_questions.research,
95
interactions.planed_text_questions.tests,
95
interactions.planed_text_questions.urls, 96
interactions.planed_text_questions.utils,
96
interactions.planed_text_questions.views,
96
interactions.report_problem, 105
interactions.report_problem.apps, 105
interactions.report_problem.models, 105
interactions.report_problem.tests, 105
interactions.report_problem.urls, 105
interactions.report_problem.views, 105
interactions.simple, 105
interactions.simple.analysis, 105
interactions.simple.apps, 106
interactions.simple.models, 106
interactions.simple.tests, 106
interactions.simple.urls, 106
interactions.simple.views, 106
interactions.solve_captcha, 107
interactions.solve_captcha.analysis, 107
interactions.solve_captcha.apps, 107
interactions.solve_captcha.forms, 107
interactions.solve_captcha.models, 108
interactions.solve_captcha.tests, 108
interactions.solve_captcha.urls, 108
interactions.solve_captcha.views, 108
interactions.student, 72
interactions.teacher, 73
interactions.tests, 74
interactions.translation, 74
interactions.urls, 74
interactions.utils, 74
interactions.views, 75

|

LIVE, 59
LIVE.settings, 59
LIVE.settings_production, 59
LIVE.settings_production_dev, 59
LIVE.settings_vagrant_lampy, 59
LIVE.urls, 59

m

manage, 25
manage_production, 25
manage_production_dev, 25
manage_vagrant_lampy, 25

q

questionpool, 59
questionpool.admin, 59
questionpool.apps, 60
questionpool.forms, 60
questionpool.models, 62
questionpool.tests, 63
questionpool.urls, 63
questionpool.views, 63

r

research, 108
research.admin, 108
research.apps, 108
research.models, 109
research.tests, 109
research.urls, 109
research.utils, 109
research.views, 109

A

- `add_answer()` (in module `core.views.polls`), 56
- `add_answer()` (in module `interactions.planned_questions.views`), 89
- `add_live_to_event()` (in module `core.views.events`), 50
- `add_ondemand_to_event()` (in module `core.views.events`), 50
- `add_param()` (in module `core.templatetags.urls`), 45
- `add_poll()` (in module `core.views.polls`), 57
- `add_question()` (in module `interactions.planned_questions.views`), 90
- `add_question()` (in module `interactions.planned_text_questions.views`), 96
- `add_task()` (in module `interactions.planned_programming_tasks.views`), 102
- `add_to_event()` (in module `core.views.events`), 50
- `analysis_event()` (in module `core.views.events`), 51
- `analytics()` (in module `core.context_processors`), 28
- `analyze_planned_programming_tasks()` (in module `interactions.planned_programming_tasks.research`), 101
- `analyze_planned_programming_tasks_csv()` (in module `research.views`), 109
- `analyze_planned_programming_tasks_csv_statistics()` (in module `research.views`), 109
- `analyze_planned_programming_tasks_show()` (in module `research.views`), 109
- `analyze_planned_programming_tasks_statistics()` (in module `interactions.planned_programming_tasks.research`), 101
- `analyze_planned_questions_student_answers()` (in module `interactions.planned_questions.research`), 88
- `analyze_planned_questions_student_answers_csv()` (in module `research.views`), 110
- `analyze_planned_questions_student_answers_csv_statistics()` (in module `research.views`), 110
- `analyze_planned_questions_student_answers_show()` (in module `research.views`), 110
- `analyze_planned_questions_student_answers_statistics()` (in module `interactions.planned_questions.research`), 88
- `analyze_planned_text_questions()` (in module `interactions.planned_text_questions.research`), 95
- `analyze_planned_text_questions_csv()` (in module `research.views`), 110
- `analyze_planned_text_questions_csv_statistics()` (in module `research.views`), 111
- `analyze_planned_text_questions_show()` (in module `research.views`), 111
- `analyze_planned_text_questions_statistics()` (in module `interactions.planned_text_questions.research`), 95
- `analyze_poll()` (in module `core.views.polls`), 57
- `analyze_poll_questions()` (in module `core.research`), 42
- `analyze_poll_questions_csv()` (in module `research.views`), 111
- `analyze_poll_questions_show()` (in module `research.views`), 111
- `answer_teacher_error()` (in module `interactions.views`), 75
- `answer_teacher_error_post_event()` (in module `interactions.views`), 75
- `AnswerErrorForm` (class in `interactions.forms`), 68
- `AnswerForm` (class in `interactions.ask_students.forms`), 77
- `AnswerForm` (class in `interactions.ask_teacher.forms`), 81
- `AnswerForm` (class in `interactions.planned_programming_tasks.forms`), 99
- `AnswerForm` (class in `interactions.planned_questions.forms`), 86
- `AnswerForm` (class in `interactions.planned_text_questions.forms`), 93
- `AnswerFromStudents` (class in `interactions.ask_students.models`), 78
- `AnswerFromStudents.DoesNotExist`, 78
- `AnswerFromStudents.MultipleObjectsReturned`, 78
- `AnswerOfStudent` (class in `interactions.planned_text_questions.models`), 94
- `AnswerOfStudent.DoesNotExist`, 94
- `AnswerOfStudent.MultipleObjectsReturned`, 94
- `ask()` (in module `interactions.ask_students.views`), 79

`ask()` (in module `interactions.ask_teacher.views`), 82

B

`browse_teacher_errors()` (in module `core.views.events`), 51

`build_plan()` (in module `interactions.views`), 75

`bulk_user()` (in module `core.views.accounts`), 48

C

`CallHistory` (class in `interactions.models`), 68

`CallHistory.DoesNotExist`, 68

`CallHistory.MultipleObjectsReturned`, 68

`CallHistoryTest` (class in `interactions.tests`), 74

`CaptchaForm` (class in `interactions.solve_captcha.forms`), 107

`Category` (class in `core.models`), 33

`Category.DoesNotExist`, 33

`Category.MultipleObjectsReturned`, 33

`category_list()` (in module `core.research`), 42

`category_list_csv()` (in module `research.views`), 112

`category_list_show()` (in module `research.views`), 112

`CategoryAdmin` (class in `core.admin`), 26

`CategoryTranslationOptions` (class in `core.translation`), 46

`ChoiceOfStudent` (class in `interactions.planned_questions.models`), 87

`ChoiceOfStudent.DoesNotExist`, 87

`ChoiceOfStudent.MultipleObjectsReturned`, 87

`clean()` (`core.fields.TimedeltaFormField` method), 29

`clean()` (`core.forms.LiveEventForm` method), 30

`clean()` (`core.forms.OnDemandEventForm` method), 31

`clean()` (`interactions.student.StudentInteractionLoader` method), 72

`clean()` (`interactions.teacher.TeacherInteractionLoader` method), 73

`clean_fields()` (`core.models.Category` method), 33

`clean_fields()` (`core.models.NewsEntry` method), 39

`clean_fields()` (`interactions.models.InteractionMethod` method), 71

`clean_fields()` (`interactions.models.InteractionType` method), 71

`close()` (`interactions.models.CallHistory` method), 68

`close()` (`interactions.models.Interaction` method), 70

`ConfigEntry` (class in `core.models`), 33

`ConfigEntry.DoesNotExist`, 34

`ConfigEntry.MultipleObjectsReturned`, 34

`ConfigEntryAdmin` (class in `core.admin`), 26

`confirm()` (in module `interactions.simple.views`), 106

`contact()` (in module `core.views.basic`), 49

`ContactForm` (class in `core.forms`), 30

`core`
module, 26

`core.admin`
module, 26

`core.apps`
module, 27

`core.config`
module, 27

`core.context_processors`
module, 28

`core.fields`
module, 28

`core.forms`
module, 30

`core.middleware`
module, 33

`core.models`
module, 33

`core.research`
module, 42

`core.templatetags`
module, 44

`core.templatetags.group_tags`
module, 44

`core.templatetags.urls`
module, 45

`core.tests`
module, 45

`core.translation`
module, 46

`core.urls`
module, 46

`core.utils`
module, 46

`core.validators`
module, 47

`core.views`
module, 48

`core.views.accounts`
module, 48

`core.views.basic`
module, 49

`core.views.events`
module, 50

`core.views.joined`
module, 54

`core.views.polls`
module, 56

`CoreConfig` (class in `core.apps`), 27

D

`debug_state()` (in module `core.context_processors`), 28

`delete_answer()` (in module `core.views.polls`), 57

`delete_answer()` (in module `interactions.planned_questions.views`), 90

`delete_event()` (in module `core.views.events`), 51

`delete_poll()` (in module `core.views.polls`), 57

`delete_question()` (in module `interactions.planned_programming_tasks.views`),

- 102
- `delete_question()` (in module *interactions.planed_questions.views*), 90
- `delete_question()` (in module *interactions.planed_text_questions.views*), 96
- `details_event()` (in module *core.views.events*), 51
- `detect()` (*core.utils.PlayerDetector* method), 46
- ## E
- `edit_event_metadata()` (in module *core.views.events*), 52
- `edit_user()` (in module *core.views.accounts*), 48
- `error_lookup()` (in module *interactions.teacher*), 73
- ErrorToTeacher* (class in *interactions.models*), 69
- ErrorToTeacher.DoesNotExist*, 70
- ErrorToTeacher.MultipleObjectsReturned*, 70
- Event* (class in *core.models*), 34
- Event.DoesNotExist*, 34
- Event.MultipleObjectsReturned*, 34
- EventAdmin* (class in *core.admin*), 26
- EventForm* (class in *core.forms*), 30
- EventTests* (class in *core.tests*), 45
- `extern_embed_login()` (in module *core.views.accounts*), 48
- ## F
- `format_student_lookup()` (in module *core.utils*), 46
- `format_teacher_lookup()` (in module *core.utils*), 46
- `formfield()` (*core.fields.TimedeltaField* method), 28
- `formfield()` (*core.fields.URLModelField* method), 29
- `from_db_value()` (*core.fields.TimedeltaField* method), 28
- ## G
- `generate_pool_question()` (in module *question-pool.views*), 63
- `get_absolute_end()` (*interactions.models.CallHistory* method), 68
- `get_absolute_length()` (*core.models.History* method), 35
- `get_all_ims_for_mode()` (*core.models.JoinedUser* method), 36
- `get_answers()` (in module *interactions.planed_questions.utils*), 89
- `get_at_relative()` (in module *interactions.utils*), 74
- `get_attention_level()` (*core.models.JoinedUser* method), 36
- `get_attention_level()` (*core.models.LiveEvent* method), 38
- `get_attention_level()` (in module *interactions.ask_students.analysis*), 77
- `get_attention_level()` (in module *interactions.ask_teacher.analysis*), 80
- `get_attention_level()` (in module *interactions.attention.analysis*), 84
- `get_attention_level()` (in module *interactions.planed_programming_tasks.analysis*), 98
- `get_attention_level()` (in module *interactions.planed_questions.analysis*), 86
- `get_attention_level()` (in module *interactions.planed_text_questions.analysis*), 93
- `get_attention_level()` (in module *interactions.simple.analysis*), 105
- `get_attention_level()` (in module *interactions.solve_captcha.analysis*), 107
- `get_attention_level_per_history()` (in module *interactions.ask_students.analysis*), 77
- `get_attention_level_per_history()` (in module *interactions.ask_teacher.analysis*), 80
- `get_attention_level_per_history()` (in module *interactions.attention.analysis*), 84
- `get_attention_level_per_history()` (in module *interactions.planed_programming_tasks.analysis*), 99
- `get_attention_level_per_history()` (in module *interactions.planed_questions.analysis*), 86
- `get_attention_level_per_history()` (in module *interactions.planed_text_questions.analysis*), 93
- `get_attention_level_per_history()` (in module *interactions.simple.analysis*), 106
- `get_attention_level_per_history()` (in module *interactions.solve_captcha.analysis*), 107
- `get_avg_joined()` (*core.models.JoinedUser* method), 36
- `get_by_py_path()` (in module *core.utils*), 46
- `get_col_from_data_list()` (in module *re-search.utils*), 109
- `get_config_bool()` (in module *core.config*), 27
- `get_config_bytes()` (in module *core.config*), 27
- `get_config_int()` (in module *core.config*), 27
- `get_config_string()` (in module *core.config*), 27
- `get_configs()` (in module *core.config*), 28
- `get_current_attention_level()` (*interactions.models.CallHistory* method), 68
- `get_current_time_based_attention_level()` (in module *interactions.analysis*), 67
- `get_db_prep_lookup()` (*core.fields.TimedeltaField* method), 29
- `get_db_prep_save()` (*core.fields.TimedeltaField* method), 29
- `get_deferred_fields()` (*core.models.Category* method), 33
- `get_deferred_fields()` (*core.models.NewsEntry* method), 39
- `get_deferred_fields()` (*interactions.models.InteractionMethod* method), 71
- `get_deferred_fields()` (*interactions.models.InteractionType* method), 71
- `get_duration()` (*core.models.Event* method), 34
- `get_duration()` (*core.models.LiveEvent* method), 38
- `get_duration()` (*core.models.OnDemandEvent*

method), 40

get_duration() (in module *interactions.utils*), 74

get_error_return() (core.utils.PlayerDetector method), 46

get_event_duration_1q_for_mode() (core.models.JoinedUser method), 36

get_event_duration_3q_for_mode() (core.models.JoinedUser method), 36

get_event_duration_for_mode() (core.models.JoinedUser method), 36

get_event_for_mode() (core.models.JoinedUser method), 36

get_event_for_mode() (core.models.PostEventPoll method), 41

get_event_real_end() (core.models.JoinedUser method), 36

get_event_real_start() (core.models.JoinedUser method), 36

get_first_history() (core.models.JoinedUser method), 36

get_ims_for_mode() (core.models.JoinedUser method), 37

get_interactions() (in module *core.utils*), 46

get_internal_type() (core.fields.TimedeltaField method), 29

get_join_percentage() (core.models.History method), 35

get_joined_users() (core.models.Event method), 34

get_joined_users() (core.models.LiveEvent method), 38

get_joined_users() (core.models.OnDemandEvent method), 40

get_last_call() (interactions.models.Interaction method), 70

get_last_history() (core.models.JoinedUser method), 37

get_last_percentage() (core.models.History method), 35

get_latest_student_interaction() (core.models.History method), 35

get_leave_percentage() (core.models.History method), 35

get_length() (core.models.History method), 35

get_length_diff() (core.models.History method), 35

get_length_percentage() (core.models.History method), 35

get_max_joined() (core.models.JoinedUser method), 37

get_min_joined() (core.models.JoinedUser method), 37

get_next_to_last_call() (interactions.models.Interaction method), 70

get_num_correct_answers() (in module *interactions.planned_questions.utils*), 89

get_num_history() (core.models.JoinedUser method), 37

get_num_ondemand_events() (core.models.Category method), 33

get_params() (core.utils.PlayerDetector method), 46

get_player_types() (core.models.LiveEvent method), 38

get_player_types() (core.models.OnDemandEvent method), 40

get_position_in_slot() (in module *interactions.utils*), 75

get_post_event_attention_level() (core.models.History method), 35

get_post_event_attention_level() (interactions.models.CallHistory method), 69

get_post_event_time_based_attention_level() (in module *interactions.analysis*), 67

get_published_at() (core.models.OnDemandEvent method), 40

get_readable_state() (interactions.models.CallHistory method), 69

get_readable_state() (interactions.models.Interaction method), 70

get_response_delay() (interactions.models.CallHistory method), 69

get_selector() (in module *core.utils*), 46

get_settings_link() (core.models.UsedLiveIM method), 41

get_settings_link() (core.models.UsedOnDemandIM method), 42

get_state() (interactions.models.CallHistory method), 69

get_state() (interactions.models.Interaction method), 70

get_student_detail_links() (in module *interactions.ask_students.utils*), 78

get_student_detail_links() (in module *interactions.ask_teacher.utils*), 82

get_student_detail_links() (in module *interactions.planned_programming_tasks.utils*), 102

get_student_detail_links() (in module *interactions.planned_questions.utils*), 89

get_student_detail_links() (in module *interactions.planned_text_questions.utils*), 96

get_student_interactions() (core.models.History method), 35

get_sum_joined() (core.models.JoinedUser method), 37

get_teacher() (core.models.LiveEvent method), 38

get_teacher() (core.models.OnDemandEvent method), 40

get_teacher_detail_links() (core.models.UsedLiveIM method), 41

get_teacher_detail_links() (core.models.UsedOnDemandIM method), 42

get_teacher_detail_links() (in module *interactions.ask_students.utils*), 78

get_teacher_detail_links() (in module *interac-*

[tions.ask_teacher.utils](#)), 82
[get_teacher_detail_links\(\)](#) (in module [interactions.planned_programming_tasks.utils](#)), 102
[get_teacher_detail_links\(\)](#) (in module [interactions.planned_questions.utils](#)), 89
[get_teacher_detail_links\(\)](#) (in module [interactions.planned_text_questions.utils](#)), 96
[get_title\(\)](#) ([core.models.LiveEvent](#) method), 38
[get_title\(\)](#) ([core.models.OnDemandEvent](#) method), 40
[get_user_code_output\(\)](#) (in module [interactions.planned_programming_tasks.utils](#)), 102
[get_watched_timespan\(\)](#) ([core.models.JoinedUser](#) method), 37
[get_watched_timespan_percent\(\)](#) ([core.models.JoinedUser](#) method), 37
[GroupCheckNode](#) (class in [core.templatetags.group_tags](#)), 44

H

[has_add_permission\(\)](#) ([interactions.planned_programming_tasks.admin.PlannedProgrammingTasksToStudentsAdmin](#) method), 98
[has_add_permission\(\)](#) ([interactions.planned_questions.admin.PlannedQuestionToStudentsAdmin](#) method), 85
[has_add_permission\(\)](#) ([interactions.planned_questions.admin.PossibleAnswerInline](#) method), 85
[has_add_permission\(\)](#) ([interactions.planned_text_questions.admin.PlannedTextQuestionToStudentsAdmin](#) method), 92
[has_text\(\)](#) ([core.models.PostEventPoll](#) method), 41
[hello_interaction\(\)](#) (in module [interactions.views](#)), 76
[hello_teacher_interaction\(\)](#) (in module [interactions.views](#)), 76
[History](#) (class in [core.models](#)), 35
[History.DoesNotExist](#), 35
[History.MultipleObjectsReturned](#), 35

I

[ifusergroup\(\)](#) (in module [core.templatetags.group_tags](#)), 44
[index\(\)](#) (in module [core.views.basic](#)), 49
[index\(\)](#) (in module [interactions.simple.views](#)), 106
[index\(\)](#) (in module [interactions.solve_captcha.views](#)), 108
[Interaction](#) (class in [interactions.models](#)), 70
[Interaction.DoesNotExist](#), 70
[Interaction.MultipleObjectsReturned](#), 70
[InteractionMethod](#) (class in [interactions.models](#)), 71
[InteractionMethod.DoesNotExist](#), 71
[InteractionMethod.MultipleObjectsReturned](#), 71
[InteractionMethodAdmin](#) (class in [interactions.admin](#)), 67
[InteractionMethodTranslationOptions](#) (class in [interactions.translation](#)), 74
[interactions](#)
 module, 66
 [interactions.admin](#)
 module, 67
 [interactions.analysis](#)
 module, 67
 [interactions.apps](#)
 module, 68
 [interactions.ask_students](#)
 module, 77
 [interactions.ask_students.analysis](#)
 module, 77
 [interactions.ask_students.apps](#)
 module, 77
 [interactions.ask_students.forms](#)
 module, 77
 [interactions.ask_students.models](#)
 module, 78
 [interactions.ask_students.tests](#)
 module, 78
 [interactions.ask_students.urls](#)
 module, 78
 [interactions.ask_students.utils](#)
 module, 78
 [interactions.ask_students.views](#)
 module, 79
 [interactions.ask_teacher](#)
 module, 80
 [interactions.ask_teacher.analysis](#)
 module, 80
 [interactions.ask_teacher.apps](#)
 module, 81
 [interactions.ask_teacher.forms](#)
 module, 81
 [interactions.ask_teacher.models](#)
 module, 81
 [interactions.ask_teacher.tests](#)
 module, 82
 [interactions.ask_teacher.urls](#)
 module, 82
 [interactions.ask_teacher.utils](#)
 module, 82
 [interactions.ask_teacher.views](#)
 module, 82
 [interactions.attention](#)
 module, 84
 [interactions.attention.analysis](#)
 module, 84
 [interactions.attention.apps](#)
 module, 84
 [interactions.attention.models](#)
 module, 84
 [interactions.attention.tests](#)
 module, 85
 [interactions.attention.urls](#)
 module, 85

<code>interactions.attention.views</code>	<code>interactions.planed_text_questions.forms</code>
module, 85	module, 93
<code>interactions.forms</code>	<code>interactions.planed_text_questions.models</code>
module, 68	module, 94
<code>interactions.models</code>	<code>interactions.planed_text_questions.research</code>
module, 68	module, 95
<code>interactions.planed_programming_tasks</code>	<code>interactions.planed_text_questions.tests</code>
module, 98	module, 95
<code>interactions.planed_programming_tasks.admin</code>	<code>interactions.planed_text_questions.urls</code>
module, 98	module, 96
<code>interactions.planed_programming_tasks.analysis</code>	<code>interactions.planed_text_questions.utils</code>
module, 98	module, 96
<code>interactions.planed_programming_tasks.apps</code>	<code>interactions.planed_text_questions.views</code>
module, 99	module, 96
<code>interactions.planed_programming_tasks.forms</code>	<code>interactions.report_problem</code>
module, 99	module, 105
<code>interactions.planed_programming_tasks.models</code>	<code>interactions.report_problem.apps</code>
module, 100	module, 105
<code>interactions.planed_programming_tasks.research</code>	<code>interactions.report_problem.models</code>
module, 101	module, 105
<code>interactions.planed_programming_tasks.tests</code>	<code>interactions.report_problem.tests</code>
module, 102	module, 105
<code>interactions.planed_programming_tasks.urls</code>	<code>interactions.report_problem.urls</code>
module, 102	module, 105
<code>interactions.planed_programming_tasks.utils</code>	<code>interactions.report_problem.views</code>
module, 102	module, 105
<code>interactions.planed_programming_tasks.views</code>	<code>interactions.simple</code>
module, 102	module, 105
<code>interactions.planed_questions</code>	<code>interactions.simple.analysis</code>
module, 85	module, 105
<code>interactions.planed_questions.admin</code>	<code>interactions.simple.apps</code>
module, 85	module, 106
<code>interactions.planed_questions.analysis</code>	<code>interactions.simple.models</code>
module, 86	module, 106
<code>interactions.planed_questions.apps</code>	<code>interactions.simple.tests</code>
module, 86	module, 106
<code>interactions.planed_questions.forms</code>	<code>interactions.simple.urls</code>
module, 86	module, 106
<code>interactions.planed_questions.models</code>	<code>interactions.simple.views</code>
module, 87	module, 106
<code>interactions.planed_questions.research</code>	<code>interactions.solve_captcha</code>
module, 88	module, 107
<code>interactions.planed_questions.tests</code>	<code>interactions.solve_captcha.analysis</code>
module, 89	module, 107
<code>interactions.planed_questions.urls</code>	<code>interactions.solve_captcha.apps</code>
module, 89	module, 107
<code>interactions.planed_questions.utils</code>	<code>interactions.solve_captcha.forms</code>
module, 89	module, 107
<code>interactions.planed_questions.views</code>	<code>interactions.solve_captcha.models</code>
module, 89	module, 108
<code>interactions.planed_text_questions</code>	<code>interactions.solve_captcha.tests</code>
module, 92	module, 108
<code>interactions.planed_text_questions.admin</code>	<code>interactions.solve_captcha.urls</code>
module, 92	module, 108
<code>interactions.planed_text_questions.analysis</code>	<code>interactions.solve_captcha.views</code>
module, 93	module, 108
<code>interactions.planed_text_questions.apps</code>	<code>interactions.student</code>
module, 93	module, 72

[interactions.teacher module](#), 73
[interactions.tests module](#), 74
[interactions.translation module](#), 74
[interactions.urls module](#), 74
[interactions.utils module](#), 74
[interactions.views module](#), 75
[InteractionsAskStudentsConfig](#) (class in [interactions.ask_students.apps](#)), 77
[InteractionsAskTeacherConfig](#) (class in [interactions.ask_teacher.apps](#)), 81
[InteractionsAttentionConfig](#) (class in [interactions.attention.apps](#)), 84
[InteractionsConfig](#) (class in [interactions.apps](#)), 68
[InteractionsPlannedProgrammingTasksConfig](#) (class in [interactions.planned_programming_tasks.apps](#)), 99
[InteractionsPlannedQuestionsConfig](#) (class in [interactions.planned_questions.apps](#)), 86
[InteractionsPlannedTextQuestionsConfig](#) (class in [interactions.planned_text_questions.apps](#)), 93
[InteractionsReportProblemConfig](#) (class in [interactions.report_problem.apps](#)), 105
[InteractionsSimpleConfig](#) (class in [interactions.simple.apps](#)), 106
[InteractionsSolveCaptchaConfig](#) (class in [interactions.solve_captcha.apps](#)), 107
[InteractionType](#) (class in [interactions.models](#)), 71
[InteractionType.DoesNotExist](#), 71
[InteractionType.MultipleObjectsReturned](#), 71
[InteractionTypeAdmin](#) (class in [interactions.admin](#)), 67
[InteractionTypeTranslationOptions](#) (class in [interactions.translation](#)), 74
[is_editable\(\)](#) ([core.models.PostEventPoll](#) method), 41
[is_here\(\)](#) ([core.models.JoinedUser](#) method), 37
[is_in_researcher_group\(\)](#) (in module [core.utils](#)), 46
[is_in_teacher_group\(\)](#) (in module [core.utils](#)), 47
[is_joinable\(\)](#) ([core.models.Event](#) method), 34
[is_live\(\)](#) ([core.models.Event](#) method), 34
[is_ondemand\(\)](#) ([core.models.Event](#) method), 34
[is_ready\(\)](#) ([core.models.Event](#) method), 34
[is_ready\(\)](#) ([core.models.LiveEvent](#) method), 38
[is_ready\(\)](#) ([core.models.OnDemandEvent](#) method), 40
[is_started_event\(\)](#) (in module [core.views.joined](#)), 54
[is_too_thight\(\)](#) (in module [interactions.utils](#)), 75
[is_user_joined\(\)](#) ([core.models.Event](#) method), 34

[is_valid\(\)](#) ([interactions.student.StudentInteractionLoader](#) method), 72
[is_valid\(\)](#) ([interactions.teacher.TeacherInteractionLoader](#) method), 73
[is_watchable\(\)](#) ([core.models.Event](#) method), 35

J

[join\(\)](#) ([core.models.JoinedUser](#) method), 37
[join_event\(\)](#) (in module [core.views.joined](#)), 54
[JoinedUser](#) (class in [core.models](#)), 36
[JoinedUser.DoesNotExist](#), 36
[JoinedUser.MultipleObjectsReturned](#), 36
[JoinedUserTests](#) (class in [core.tests](#)), 45

L

[leave\(\)](#) ([core.models.JoinedUser](#) method), 37
[leave_event\(\)](#) (in module [core.views.joined](#)), 54
[leave_handler\(\)](#) (in module [core.views.joined](#)), 54
[list_planned_programming_tasks\(\)](#) (in module [interactions.planned_programming_tasks.research](#)), 101
[list_planned_programming_tasks_answers\(\)](#) (in module [interactions.planned_programming_tasks.research](#)), 101
[list_planned_programming_tasks_answers_csv\(\)](#) (in module [research.views](#)), 112
[list_planned_programming_tasks_answers_show\(\)](#) (in module [research.views](#)), 112
[list_planned_programming_tasks_csv\(\)](#) (in module [research.views](#)), 113
[list_planned_programming_tasks_show\(\)](#) (in module [research.views](#)), 113
[list_planned_questions\(\)](#) (in module [interactions.planned_questions.research](#)), 88
[list_planned_questions_answers\(\)](#) (in module [interactions.planned_questions.research](#)), 88
[list_planned_questions_answers_csv\(\)](#) (in module [research.views](#)), 113
[list_planned_questions_csv\(\)](#) (in module [research.views](#)), 113
[list_planned_questions_show\(\)](#) (in module [research.views](#)), 114
[list_planned_questions_student_answers\(\)](#) (in module [interactions.planned_questions.research](#)), 88
[list_planned_questions_student_answers_csv\(\)](#) (in module [research.views](#)), 114
[list_planned_questions_student_answers_show\(\)](#) (in module [research.views](#)), 114
[list_planned_text_questions\(\)](#) (in module [interactions.planned_text_questions.research](#)), 95
[list_planned_text_questions_answers\(\)](#) (in module [interac-](#)

[tions.planed_text_questions.research](#)),
[95](#)
[list_planed_text_questions_answers_csv\(\)](#)
 (in module [research.views](#)), [114](#)
[list_planed_text_questions_answers_show\(\)](#)
 (in module [research.views](#)), [114](#)
[list_planed_text_questions_csv\(\)](#) (in module
[research.views](#)), [115](#)
[list_planed_text_questions_show\(\)](#) (in module
[research.views](#)), [115](#)
[list_poll_possible_answers\(\)](#) (in module
[core.research](#)), [42](#)
[list_poll_possible_answers_csv\(\)](#) (in module
[research.views](#)), [115](#)
[list_poll_questions\(\)](#) (in module [core.research](#)),
[42](#)
[list_poll_questions_csv\(\)](#) (in module [re-](#)
[search.views](#)), [115](#)
[list_poll_questions_show\(\)](#) (in module [re-](#)
[search.views](#)), [116](#)
[list_poll_student_answers\(\)](#) (in module
[core.research](#)), [43](#)
[list_poll_student_answers_csv\(\)](#) (in module [re-](#)
[search.views](#)), [116](#)
[list_poll_student_answers_show\(\)](#) (in module
[research.views](#)), [116](#)
 LIVE
 module, [59](#)
 LIVE.settings
 module, [59](#)
 LIVE.settings_production
 module, [59](#)
 LIVE.settings_production_dev
 module, [59](#)
 LIVE.settings_vagrant_lampy
 module, [59](#)
 LIVE.urls
 module, [59](#)
[live_event_list\(\)](#) (in module [core.research](#)), [43](#)
[live_event_list_csv\(\)](#) (in module [research.views](#)),
[116](#)
[live_event_list_show\(\)](#) (in module [re-](#)
[search.views](#)), [117](#)
[LiveEvent](#) (class in [core.models](#)), [37](#)
[LiveEvent.DoesNotExist](#), [38](#)
[LiveEvent.MultipleObjectsReturned](#), [38](#)
[LiveEventAdmin](#) (class in [core.admin](#)), [26](#)
[LiveEventForm](#) (class in [core.forms](#)), [30](#)
[load\(\)](#) ([interactions.student.StudentInteractionLoader](#)
 method), [72](#)
[load\(\)](#) ([interactions.teacher.TeacherInteractionLoader](#)
 method), [73](#)
[LocalInfo](#) (class in [core.models](#)), [38](#)
[LocalInfo.DoesNotExist](#), [38](#)
[LocalInfo.MultipleObjectsReturned](#), [38](#)
[LocalInfoInline](#) (class in [core.admin](#)), [26](#)
[lookup_student\(\)](#) (in module [core.views.joined](#)), [54](#)
[lookup_teacher\(\)](#) (in module [core.views.joined](#)), [55](#)

[lookupTest](#) (class in [interactions.tests](#)), [74](#)
[lti_after_user_creation\(\)](#) (in module [core.utils](#)),
[47](#)

M

manage
 module, [25](#)
[manage_events\(\)](#) (in module [core.views.events](#)), [52](#)
[manage_polls\(\)](#) (in module [core.views.polls](#)), [58](#)
[manage_production](#)
 module, [25](#)
[manage_production_dev](#)
 module, [25](#)
[manage_vagrant_lampy](#)
 module, [25](#)
[mark_ready\(\)](#) (in module [interac-](#)
[tions.planed_programming_tasks.views](#)),
[103](#)
[mark_ready\(\)](#) (in module [interac-](#)
[tions.planed_questions.views](#)), [91](#)
[mark_ready\(\)](#) (in module [interac-](#)
[tions.planed_text_questions.views](#)), [97](#)
[media](#) ([core.forms.ContactForm](#) property), [30](#)
[media](#) ([core.forms.EventForm](#) property), [30](#)
[media](#) ([core.forms.LiveEventForm](#) property), [30](#)
[media](#) ([core.forms.NewsCommentForm](#) property), [31](#)
[media](#) ([core.forms.OnDemandEventForm](#) property), [31](#)
[media](#) ([core.forms.PEPAnswerForm](#) property), [31](#)
[media](#) ([core.forms.PEPAnswerTextForm](#) property), [31](#)
[media](#) ([core.forms.PEPPossibleAnswerForm](#) property),
[32](#)
[media](#) ([core.forms.PostEventPollForm](#) property), [32](#)
[media](#) ([core.forms.UserCreateForm](#) property), [32](#)
[media](#) ([core.forms.UserCreateMultipleForm](#) property),
[32](#)
[media](#) ([core.forms.UserEditForm](#) property), [33](#)
[media](#) ([core.forms.Youtube2ClickForm](#) property), [33](#)
[media](#) ([interactions.ask_students.forms.AnswerForm](#)
 property), [77](#)
[media](#) ([interactions.ask_teacher.forms.AnswerForm](#)
 property), [81](#)
[media](#) ([interactions.forms.AnswerErrorForm](#) prop-
 erty), [68](#)
[media](#) ([interactions.planed_programming_tasks.forms.AnswerForm](#)
 property), [99](#)
[media](#) ([interactions.planed_programming_tasks.forms.PorgrammingTask](#)
 property), [100](#)
[media](#) ([interactions.planed_programming_tasks.forms.TeacherNoteForm](#)
 property), [100](#)
[media](#) ([interactions.planed_questions.forms.AnswerForm](#)
 property), [86](#)
[media](#) ([interactions.planed_questions.forms.PossibleAnswerForm](#)
 property), [86](#)
[media](#) ([interactions.planed_questions.forms.QuestionForm](#)
 property), [87](#)
[media](#) ([interactions.planed_text_questions.forms.AnswerForm](#)
 property), [93](#)

[media\(interactions.planned_text_questions.forms.TextQuestionForm property\)](#), 94
[media\(interactions.solve_captcha.forms.CaptchaForm property\)](#), 107
[media\(questionpool.forms.PoolMultipleChoiceQuestionForm property\)](#), 60
[media\(questionpool.forms.PoolPossibleAnswerForm property\)](#), 60
[media\(questionpool.forms.PoolProgrammingTaskForm property\)](#), 61
[media\(questionpool.forms.PoolTextQuestionForm property\)](#), 61
[media\(questionpool.forms.QuestionpoolForm property\)](#), 61
[messages_authenticated\(\)](#) (in module [core.context_processors](#)), 28
[messages_unauthenticated\(\)](#) (in module [core.context_processors](#)), 28
[mode_to_type\(\)](#) (in module [core.utils](#)), 47
[model\(core.admin.LocalInfoInline attribute\)](#), 26
[model\(core.admin.NewsCommentInline attribute\)](#), 26
[model\(core.admin.UsedLiveIMInline attribute\)](#), 27
[model\(core.admin.UsedOnDemandIMInline attribute\)](#), 27
[model\(core.admin.UserInfoInline attribute\)](#), 27
[model\(interactions.planned_questions.admin.PossibleAnswerInline attribute\)](#), 85
[model\(questionpool.admin.PoolMultipleChoiceQuestionInline attribute\)](#), 59
[model\(questionpool.admin.PoolProgrammingTaskInline attribute\)](#), 59
[model\(questionpool.admin.PoolTextQuestionInline attribute\)](#), 59
[module](#)
 [core](#), 26
 [core.admin](#), 26
 [core.apps](#), 27
 [core.config](#), 27
 [core.context_processors](#), 28
 [core.fields](#), 28
 [core.forms](#), 30
 [core.middleware](#), 33
 [core.models](#), 33
 [core.research](#), 42
 [core.templatetags](#), 44
 [core.templatetags.group_tags](#), 44
 [core.templatetags.urls](#), 45
 [core.tests](#), 45
 [core.translation](#), 46
 [core.urls](#), 46
 [core.utils](#), 46
 [core.validators](#), 47
 [core.views](#), 48
 [core.views.accounts](#), 48
 [core.views.basic](#), 49
 [core.views.events](#), 50
 [core.views.joined](#), 54
 [core.views.polls](#), 56
[interactions](#), 66
 [interactions.admin](#), 67
 [interactions.analysis](#), 67
 [interactions.apps](#), 68
 [interactions.ask_students](#), 77
 [interactions.ask_students.analysis](#), 77
 [interactions.ask_students.apps](#), 77
 [interactions.ask_students.forms](#), 77
 [interactions.ask_students.models](#), 78
 [interactions.ask_students.tests](#), 78
 [interactions.ask_students.urls](#), 78
 [interactions.ask_students.utils](#), 78
 [interactions.ask_students.views](#), 79
 [interactions.ask_teacher](#), 80
 [interactions.ask_teacher.analysis](#), 80
 [interactions.ask_teacher.apps](#), 81
 [interactions.ask_teacher.forms](#), 81
 [interactions.ask_teacher.models](#), 81
 [interactions.ask_teacher.tests](#), 82
 [interactions.ask_teacher.urls](#), 82
 [interactions.ask_teacher.utils](#), 82
 [interactions.ask_teacher.views](#), 82
 [interactions.attention](#), 84
 [interactions.attention.analysis](#), 84
 [interactions.attention.apps](#), 84
 [interactions.attention.models](#), 84
 [interactions.attention.tests](#), 85
 [interactions.attention.urls](#), 85
 [interactions.attention.views](#), 85
 [interactions.forms](#), 68
 [interactions.models](#), 68
 [interactions.planned_programming_tasks](#), 98
 [interactions.planned_programming_tasks.admin](#), 98
 [interactions.planned_programming_tasks.analysis](#), 98
 [interactions.planned_programming_tasks.apps](#), 99
 [interactions.planned_programming_tasks.forms](#), 99
 [interactions.planned_programming_tasks.models](#), 100
 [interactions.planned_programming_tasks.research](#), 101
 [interactions.planned_programming_tasks.tests](#), 102
 [interactions.planned_programming_tasks.urls](#), 102
 [interactions.planned_programming_tasks.utils](#), 102
 [interactions.planned_programming_tasks.views](#), 102
 [interactions.planned_questions](#), 85
 [interactions.planned_questions.admin](#), 85
 [interactions.planned_questions.analysis](#), 86
 [interactions.planned_questions.apps](#), 86

interactions.planed_questions.forms, 86
interactions.planed_questions.models, 87
interactions.planed_questions.research, 88
interactions.planed_questions.tests, 89
interactions.planed_questions.urls, 89
interactions.planed_questions.utils, 89
interactions.planed_questions.views, 89
interactions.planed_text_questions, 92
interactions.planed_text_questions.admin, 92
interactions.planed_text_questions.analysis, 93
interactions.planed_text_questions.apps, 93
interactions.planed_text_questions.forms, 93
interactions.planed_text_questions.models, 94
interactions.planed_text_questions.research, 95
interactions.planed_text_questions.tests, 95
interactions.planed_text_questions.urls, 96
interactions.planed_text_questions.utils, 96
interactions.planed_text_questions.views, 96
interactions.report_problem, 105
interactions.report_problem.apps, 105
interactions.report_problem.models, 105
interactions.report_problem.tests, 105
interactions.report_problem.urls, 105
interactions.report_problem.views, 105
interactions.simple, 105
interactions.simple.analysis, 105
interactions.simple.apps, 106
interactions.simple.models, 106
interactions.simple.tests, 106
interactions.simple.urls, 106
interactions.simple.views, 106
interactions.solve_captcha, 107
interactions.solve_captcha.analysis, 107
interactions.solve_captcha.apps, 107
interactions.solve_captcha.forms, 107
interactions.solve_captcha.models, 108
interactions.solve_captcha.tests, 108
interactions.solve_captcha.urls, 108
interactions.solve_captcha.views, 108
interactions.student, 72
interactions.teacher, 73
interactions.tests, 74
interactions.translation, 74
interactions.urls, 74
interactions.utils, 74
interactions.views, 75
LIVE, 59
LIVE.settings, 59
LIVE.settings_production, 59
LIVE.settings_production_dev, 59
LIVE.settings_vagrant_lampy, 59
LIVE.urls, 59
manage, 25
manage_production, 25
manage_production_dev, 25
manage_vagrant_lampy, 25
questionpool, 59
questionpool.admin, 59
questionpool.apps, 60
questionpool.forms, 60
questionpool.models, 62
questionpool.tests, 63
questionpool.urls, 63
questionpool.views, 63
research, 108
research.admin, 108
research.apps, 108
research.models, 109
research.tests, 109
research.urls, 109
research.utils, 109
research.views, 109
multiple_choice_question_answer_delete()
 (in module questionpool.views), 63
multiple_choice_question_answer_edit_or_create()
 (in module questionpool.views), 64
multiple_choice_question_delete() (in module
 questionpool.views), 64
multiple_choice_question_edit_or_create()
 (in module questionpool.views), 64
my_created_events() (in module core.views.events),
 52
my_joined_events() (in module core.views.events),
 52
my_watched_events() (in module core.views.events),
 52
myacc() (in module core.views.accounts), 48

N

news_details() (in module core.views.basic), 49
NewsComment (class in core.models), 39
NewsComment.DoesNotExist, 39
NewsComment.MultipleObjectsReturned, 39
NewsCommentForm (class in core.forms), 31
NewsCommentInline (class in core.admin), 26
NewsEntry (class in core.models), 39
NewsEntry.DoesNotExist, 39
NewsEntry.MultipleObjectsReturned, 39
NewsEntryAdmin (class in core.admin), 26
NewsEntryTranslationOptions (class in
 core.translation), 46
newuser() (in module core.views.accounts), 48

- no_interactions() (in module *interactions.views*), 76
- normalize_history() (core.models.JoinedUser method), 37
- not_done() (interactions.models.CallHistory method), 69
- not_done() (interactions.models.Interaction method), 70
- ## O
- on_demand_events() (in module *core.views.events*), 52
- ondemand_event_list() (in module *core.research*), 43
- ondemand_event_list_csv() (in module *research.views*), 117
- ondemand_event_list_show() (in module *research.views*), 117
- OnDemandEvent (class in *core.models*), 40
- OnDemandEvent.DoesNotExist, 40
- OnDemandEvent.MultipleObjectsReturned, 40
- OnDemandEventAdmin (class in *core.admin*), 26
- OnDemandEventForm (class in *core.forms*), 31
- open_call() (interactions.models.Interaction method), 70
- ## P
- paginator() (in module *core.utils*), 47
- password_change() (in module *core.views.accounts*), 49
- password_change_done() (in module *core.views.accounts*), 49
- pause_event() (in module *core.views.joined*), 55
- PEPAnswer (class in *core.models*), 40
- PEPAnswer.DoesNotExist, 40
- PEPAnswer.MultipleObjectsReturned, 40
- PEPAnswerForm (class in *core.forms*), 31
- PEPAnswerTextForm (class in *core.forms*), 31
- PEPPossibleAnswer (class in *core.models*), 40
- PEPPossibleAnswer.DoesNotExist, 41
- PEPPossibleAnswer.MultipleObjectsReturned, 41
- PEPPossibleAnswerForm (class in *core.forms*), 31
- plan_questions() (in module *interactions.planned_questions.views*), 91
- plan_questions() (in module *interactions.planned_text_questions.views*), 97
- plan_tasks() (in module *interactions.planned_programming_tasks.views*), 103
- PlannedProgrammingTasksToStudents (class in *interactions.planned_programming_tasks.models*), 100
- PlannedProgrammingTasksToStudents.DoesNotExist, 100
- PlannedProgrammingTasksToStudents.MultipleObjectsReturned, 100
- PlannedProgrammingTasksToStudentsAdmin (class in *interactions.planned_programming_tasks.admin*), 98
- PlannedQuestionToStudents (class in *interactions.planned_questions.models*), 87
- PlannedQuestionToStudents.DoesNotExist, 87
- PlannedQuestionToStudents.MultipleObjectsReturned, 87
- PlannedQuestionToStudentsAdmin (class in *interactions.planned_questions.admin*), 85
- PlannedStudentInteraction (class in *interactions.models*), 72
- PlannedStudentInteraction.DoesNotExist, 72
- PlannedStudentInteraction.MultipleObjectsReturned, 72
- PlannedTextQuestionToStudents (class in *interactions.planned_text_questions.models*), 94
- PlannedTextQuestionToStudents.DoesNotExist, 94
- PlannedTextQuestionToStudents.MultipleObjectsReturned, 94
- PlannedTextQuestionToStudentsAdmin (class in *interactions.planned_text_questions.admin*), 92
- player_set() (in module *core.views.basic*), 49
- PlayerDetector (class in *core.utils*), 46
- playerdev() (in module *core.views.basic*), 49
- Pool (class in *questionpool.models*), 62
- Pool.Category (class in *questionpool.models*), 62
- Pool.DoesNotExist, 62
- Pool.MultipleObjectsReturned, 62
- pool_delete() (in module *questionpool.views*), 64
- pool_detail() (in module *questionpool.views*), 65
- pool_edit_or_create() (in module *questionpool.views*), 65
- PoolAdmin (class in *questionpool.admin*), 59
- PoolMultipleChoiceQuestion (class in *questionpool.models*), 62
- PoolMultipleChoiceQuestion.DoesNotExist, 62
- PoolMultipleChoiceQuestion.MultipleObjectsReturned, 62
- PoolMultipleChoiceQuestionForm (class in *questionpool.forms*), 60
- PoolMultipleChoiceQuestionInline (class in *questionpool.admin*), 59
- PoolPossibleAnswer (class in *questionpool.models*), 62
- PoolPossibleAnswer.DoesNotExist, 62
- PoolPossibleAnswer.MultipleObjectsReturned, 62
- PoolPossibleAnswerForm (class in *questionpool.forms*), 60
- PoolProgrammingTask (class in *questionpool.models*), 62
- PoolProgrammingTask.DoesNotExist, 63
- PoolProgrammingTask.MultipleObjectsReturned, 63
- PoolProgrammingTaskForm (class in *question-*

pool.forms), 60
PoolProgrammingTaskInline (class in *questionpool.admin*), 59
PoolTextQuestion (class in *questionpool.models*), 63
PoolTextQuestion.DoesNotExist, 63
PoolTextQuestion.MultipleObjectsReturned, 63
PoolTextQuestionForm (class in *questionpool.forms*), 61
PoolTextQuestionInline (class in *questionpool.admin*), 59
PorgrammingTaskForm (class in *interactions.planed_programming_tasks.forms*), 99
PossibleAnswer (class in *interactions.planed_questions.models*), 87
PossibleAnswer.DoesNotExist, 88
PossibleAnswer.MultipleObjectsReturned, 88
PossibleAnswerForm (class in *interactions.planed_questions.forms*), 86
PossibleAnswerInline (class in *interactions.planed_questions.admin*), 85
PostEventPoll (class in *core.models*), 41
PostEventPoll.DoesNotExist, 41
PostEventPoll.MultipleObjectsReturned, 41
PostEventPollForm (class in *core.forms*), 32
programming_task_delete() (in module *questionpool.views*), 65
programming_task_edit_or_create() (in module *questionpool.views*), 65

Q

QuestionForm (class in *interactions.planed_questions.forms*), 87
questionpool
 module, 59
questionpool.admin
 module, 59
questionpool.apps
 module, 60
questionpool.forms
 module, 60
questionpool.models
 module, 62
questionpool.tests
 module, 63
questionpool.urls
 module, 63
questionpool.views
 module, 63
QuestionpoolConfig (class in *questionpool.apps*), 60
QuestionpoolForm (class in *questionpool.forms*), 61
QuestionToStudents (class in *interactions.ask_students.models*), 78
QuestionToStudents.DoesNotExist, 78
QuestionToStudents.MultipleObjectsReturned, 78

QuestionToTeacher (class in *interactions.ask_teacher.models*), 81
QuestionToTeacher.DoesNotExist, 81
QuestionToTeacher.MultipleObjectsReturned, 81

R

redirect_get_params() (in module *core.utils*), 47
refresh_from_db() (*core.models.Category* method), 33
refresh_from_db() (*core.models.NewsEntry* method), 39
refresh_from_db() (*interactions.models.InteractionMethod* method), 71
refresh_from_db() (*interactions.models.InteractionType* method), 71
registration_csv() (in module *research.views*), 117
registration_show() (in module *research.views*), 117
registration_statistic() (in module *core.research*), 43
render() (*core.fields.TimedeltaWidget* method), 29
render() (*core.templatetags.group_tags.GroupCheckNode* method), 44
report() (in module *interactions.report_problem.views*), 105
research
 module, 108
research.admin
 module, 108
research.apps
 module, 108
research.models
 module, 109
research.tests
 module, 109
research.urls
 module, 109
research.utils
 module, 109
research.views
 module, 109
ResearchConfig (class in *research.apps*), 108
respond() (*interactions.models.CallHistory* method), 69
respond() (*interactions.models.Interaction* method), 70
respond() (*interactions.student.StudentInteractionLoader* method), 72
respond() (*interactions.teacher.TeacherInteractionLoader* method), 73
resume_event() (in module *core.views.joined*), 55

S

search_event_autocomplete() (in module *core.views.events*), 53

- [seek_event\(\)](#) (in module `core.views.joined`), 55
[select\(\)](#) (in module `questionpool.views`), 66
[select\(\)](#) (in module `research.views`), 118
[set_attention\(\)](#) (in module `interactions.attention.views`), 85
[SetAttention](#) (class in `interactions.attention.models`), 84
[SetAttention.DoesNotExist](#), 84
[SetAttention.MultipleObjectsReturned](#), 84
[setUp\(\)](#) (`core.tests.EventTests` method), 45
[setUp\(\)](#) (`core.tests.JoinedUserTests` method), 45
[setUp\(\)](#) (`interactions.tests.lookupTest` method), 74
[show\(\)](#) (`interactions.models.CallHistory` method), 69
[show\(\)](#) (`interactions.models.Interaction` method), 70
[show\(\)](#) (`interactions.student.StudentInteractionLoader` method), 73
[show\(\)](#) (`interactions.teacher.TeacherInteractionLoader` method), 73
[show_error_answer\(\)](#) (in module `interactions.views`), 76
[show_poll\(\)](#) (in module `core.views.polls`), 58
[show_to_student\(\)](#) (in module `core.views.joined`), 55
[show_to_teacher\(\)](#) (in module `core.views.joined`), 56
[split_seconds\(\)](#) (in module `core.fields`), 29
[start_event\(\)](#) (in module `core.views.joined`), 56
[stop_event\(\)](#) (in module `core.views.joined`), 56
[student_answer\(\)](#) (in module `interactions.ask_students.views`), 79
[student_answer\(\)](#) (in module `interactions.ask_teacher.views`), 82
[student_details\(\)](#) (in module `interactions.ask_students.views`), 79
[student_details\(\)](#) (in module `interactions.ask_teacher.views`), 82
[student_details\(\)](#) (in module `interactions.planned_programming_tasks.views`), 103
[student_details\(\)](#) (in module `interactions.planned_text_questions.views`), 97
[student_list_my\(\)](#) (in module `interactions.planned_questions.views`), 91
[student_lookup\(\)](#) (in module `interactions.student`), 73
[student_question\(\)](#) (in module `interactions.ask_students.views`), 79
[student_question\(\)](#) (in module `interactions.ask_teacher.views`), 83
[student_question\(\)](#) (in module `interactions.planned_questions.views`), 91
[student_question\(\)](#) (in module `interactions.planned_text_questions.views`), 97
[student_statistics\(\)](#) (in module `interactions.planned_questions.views`), 91
[student_task\(\)](#) (in module `interactions.planned_programming_tasks.views`), 103
[StudentInteraction](#) (class in `interactions.models`), 72
[StudentInteraction.DoesNotExist](#), 72
[StudentInteraction.MultipleObjectsReturned](#), 72
[StudentInteractionLoader](#) (class in `interactions.student`), 72
[switch_poll_state\(\)](#) (in module `core.views.polls`), 58
- ## T
- [TaskAnswerOfStudent](#) (class in `interactions.planned_programming_tasks.models`), 100
[TaskAnswerOfStudent.DoesNotExist](#), 101
[TaskAnswerOfStudent.MultipleObjectsReturned](#), 101
[teacher_answer\(\)](#) (in module `interactions.ask_students.views`), 79
[teacher_answer\(\)](#) (in module `interactions.ask_teacher.views`), 83
[teacher_answers\(\)](#) (in module `interactions.planned_programming_tasks.views`), 104
[teacher_answers\(\)](#) (in module `interactions.planned_questions.views`), 92
[teacher_answers\(\)](#) (in module `interactions.planned_text_questions.views`), 97
[teacher_details\(\)](#) (in module `interactions.ask_students.views`), 80
[teacher_details\(\)](#) (in module `interactions.ask_teacher.views`), 83
[teacher_details\(\)](#) (in module `interactions.planned_programming_tasks.views`), 104
[teacher_details\(\)](#) (in module `interactions.planned_questions.views`), 92
[teacher_details\(\)](#) (in module `interactions.planned_text_questions.views`), 98
[teacher_lookup\(\)](#) (in module `interactions.teacher`), 73
[teacher_question\(\)](#) (in module `interactions.ask_students.views`), 80
[teacher_question\(\)](#) (in module `interactions.ask_teacher.views`), 83
[TeacherInteraction](#) (class in `interactions.models`), 72
[TeacherInteraction.DoesNotExist](#), 72
[TeacherInteraction.MultipleObjectsReturned](#), 72
[TeacherInteractionLoader](#) (class in `interactions.teacher`), 73
[TeacherNoteForm](#) (class in `interactions.planned_programming_tasks.forms`), 100
[team\(\)](#) (in module `core.views.basic`), 50
[test_event_real_end\(\)](#) (`core.tests.JoinedUserTests` method), 45

[test_event_real_start\(\)](#) (*core.tests.JoinedUserTests method*), 45
[test_is_here\(\)](#) (*core.tests.JoinedUserTests method*), 45
[test_is_joinable\(\)](#) (*core.tests.EventTests method*), 45
[test_is_live\(\)](#) (*core.tests.EventTests method*), 45
[test_is_ondemand\(\)](#) (*core.tests.EventTests method*), 45
[test_is_ondemand_and_is_live\(\)](#) (*core.tests.EventTests method*), 45
[test_is_ready\(\)](#) (*core.tests.EventTests method*), 45
[test_is_user_joined\(\)](#) (*core.tests.EventTests method*), 45
[test_model\(\)](#) (*core.tests.JoinedUserTests method*), 45
[test_player\(\)](#) (*in module core.views.events*), 53
[test_state\(\)](#) (*interactions.tests.CallHistoryTest method*), 74
[test_state\(\)](#) (*interactions.tests.lookupTest method*), 74
[test_student_loader\(\)](#) (*interactions.tests.lookupTest method*), 74
[test_student_lookup\(\)](#) (*interactions.tests.lookupTest method*), 74
[text_question_delete\(\)](#) (*in module questionpool.views*), 66
[text_question_edit_or_create\(\)](#) (*in module questionpool.views*), 66
[TextQuestionForm](#) (*class in interactions.planned_text_questions.forms*), 93
[time_based_attention_level\(\)](#) (*in module interactions.analysis*), 67
[TimedeltaField](#) (*class in core.fields*), 28
[TimedeltaFormField](#) (*class in core.fields*), 29
[TimedeltaWidget](#) (*class in core.fields*), 29
[timeline_overall\(\)](#) (*in module core.views.events*), 53
[TimezoneMiddleware](#) (*class in core.middleware*), 33
[to_python\(\)](#) (*core.fields.TimedeltaField method*), 29
[to_python\(\)](#) (*core.fields.URLFormField method*), 29
[type_to_mode\(\)](#) (*in module core.utils*), 47

U

[URLFormField](#) (*class in core.fields*), 29
[URLModelField](#) (*class in core.fields*), 29
[URLValidator](#) (*class in core.validators*), 47
[UsedLiveIM](#) (*class in core.models*), 41
[UsedLiveIM.DoesNotExist](#), 41
[UsedLiveIM.MultipleObjectsReturned](#), 41
[UsedLiveIMInline](#) (*class in core.admin*), 27
[UsedOnDemandIM](#) (*class in core.models*), 41
[UsedOnDemandIM.DoesNotExist](#), 42
[UsedOnDemandIM.MultipleObjectsReturned](#), 42
[UsedOnDemandIMInline](#) (*class in core.admin*), 27
[user_csv\(\)](#) (*in module research.views*), 118
[user_show\(\)](#) (*in module research.views*), 118
[user_statistic\(\)](#) (*in module core.research*), 43

[UserAdmin](#) (*class in core.admin*), 27
[UserCreateForm](#) (*class in core.forms*), 32
[UserCreateMultipleForm](#) (*class in core.forms*), 32
[UserEditForm](#) (*class in core.forms*), 32
[UserInfo](#) (*class in core.models*), 42
[UserInfo.DoesNotExist](#), 42
[UserInfo.MultipleObjectsReturned](#), 42
[UserInfoInline](#) (*class in core.admin*), 27
[users_per_second\(\)](#) (*in module core.research*), 43
[users_per_second_csv\(\)](#) (*in module research.views*), 118
[users_per_second_csv_statistics\(\)](#) (*in module research.views*), 118
[users_per_second_show\(\)](#) (*in module research.views*), 119
[users_per_second_statistics\(\)](#) (*in module core.research*), 43

V

[value_from_datadict\(\)](#) (*core.fields.TimedeltaWidget method*), 29
[value_to_string\(\)](#) (*core.fields.TimedeltaField method*), 29
[version\(\)](#) (*in module core.context_processors*), 28
[visible_comments\(\)](#) (*core.models.NewsEntry method*), 39

W

[watch_statistic_per_user\(\)](#) (*in module core.research*), 43
[watch_statistic_per_user_csv\(\)](#) (*in module research.views*), 119
[watch_statistic_per_user_csv_statistics\(\)](#) (*in module research.views*), 119
[watch_statistic_per_user_show\(\)](#) (*in module research.views*), 119
[watch_statistic_per_user_statistics\(\)](#) (*in module core.research*), 44
[watched_timespans_per_user\(\)](#) (*in module core.research*), 44
[watched_timespans_per_user_csv\(\)](#) (*in module research.views*), 120
[watched_timespans_per_user_show\(\)](#) (*in module research.views*), 120

Y

[youtube2click\(\)](#) (*in module core.views.basic*), 50
[Youtube2ClickForm](#) (*class in core.forms*), 33
[Youtube2ClickMiddleware](#) (*class in core.middleware*), 33