## 学生端功能路由

**Lms/urls.py**

from django.conf import settings

from django.conf.urls import patterns, include, url

from ratelimitbackend import admin

from django.conf.urls.static import static

import django.contrib.auth.views

# Uncomment the next two lines to enable the admin:

if settings.DEBUG or settings.FEATURES.get('ENABLE\_DJANGO\_ADMIN\_SITE'):

admin.autodiscover()

urlpatterns = ('', # nopep8

# certificate view

url(r'^update\_certificate$', 'certificates.views.update\_certificate'),

url(r'^$', 'branding.views.index', name="root"), # Main marketing page, or redirect to courseware

url(r'^dashboard$', 'student.views.dashboard', name="dashboard"),

url(r'^token$', 'student.views.token', name="token"),

url(r'^login$', 'student.views.signin\_user', name="signin\_user"),

url(r'^register$', 'student.views.register\_user', name="register\_user"),

url(r'^auth\_user$', 'student.views.auth\_user'),

url(r'^admin\_dashboard$', 'dashboard.views.dashboard'),

# API

url(r'^api/oauth2/', include('provider.oauth2.urls', namespace='oauth2')),

url(r'^api/', include('rest\_framework.urls', namespace='rest\_framework')),

url(r'^api/', include('api.urls', namespace='api')),

url(r'^change\_email$', 'student.views.change\_email\_request', name="change\_email"),

url(r'^email\_confirm/(?P<key>[^/]\*)$', 'student.views.confirm\_email\_change'),

url(r'^change\_name$', 'student.views.change\_name\_request', name="change\_name"),

url(r'^accept\_name\_change$', 'student.views.accept\_name\_change'),

url(r'^reject\_name\_change$', 'student.views.reject\_name\_change'),

url(r'^pending\_name\_changes$', 'student.views.pending\_name\_changes'),

url(r'^event$', 'track.views.user\_track'),

url(r'^about$', 'static\_template\_view.views.render', {'template': 'about.html'}),

url(r'^download$', 'static\_template\_view.views.render', {'template': 'download.html'}),

url(r'^course-intro$', 'static\_template\_view.views.render', {'template': 'course-intro.html'}),

url(r'^t/(?P<template>[^/]\*)$', 'static\_template\_view.views.index'), # TODO: Is this used anymore? What is STATIC\_GRAB?

# for credential

url(r'^apply\_credential/(?P<course\_id>[^/]+/[^/]+/[^/]+)$', 'credential.views.apply\_credential'),

url(r'^get\_credential/(?P<course\_id>[^/]+/[^/]+/[^/]+)$', 'credential.views.get\_credential'),

url(r'^download\_credential/(?P<credential\_id>[^/]+).pdf$', 'credential.views.download\_credential'),

url(r'^image\_credential/(?P<credential\_id>[^/]+).jpg$', 'credential.views.image\_credential'),

url(r'^check\_credential/(?P<credential\_id>[^/]+)$', 'credential.views.check\_credential'),

url(r'^accounts/login$', 'student.views.accounts\_login', name="accounts\_login"),

url(r'^accounts/manage\_user\_standing', 'student.views.manage\_user\_standing',

name='manage\_user\_standing'),

url(r'^accounts/disable\_account\_ajax$', 'student.views.disable\_account\_ajax',

name="disable\_account\_ajax"),

url(r'^login\_ajax$', 'student.views.login\_user', name="login"),

url(r'^login\_ajax/(?P<error>[^/]\*)$', 'student.views.login\_user'),

url(r'^logout$', 'student.views.logout\_user', name='logout'),

url(r'^check\_email\_available$', 'student.views.check\_email\_available'),

url(r'^check\_username\_available$', 'student.views.check\_username\_available'),

url(r'^create\_account$', 'student.views.create\_account', name='create\_account'),

url(r'^activate/(?P<key>[^/]\*)$', 'student.views.activate\_account', name="activate"),

url(r'^password\_reset/$', 'student.views.password\_reset', name='password\_reset'),

## Obsolete Django views for password resets

## TODO: Replace with Mako-ized views

url(r'^password\_change/$', django.contrib.auth.views.password\_change,

name='auth\_password\_change'),

url(r'^password\_change\_done/$', django.contrib.auth.views.password\_change\_done,

name='auth\_password\_change\_done'),

url(r'^password\_reset\_confirm/(?P<uidb36>[0-9A-Za-z]+)-(?P<token>.+)/$',

'student.views.password\_reset\_confirm\_wrapper',

name='auth\_password\_reset\_confirm'),

url(r'^password\_reset\_complete/$', django.contrib.auth.views.password\_reset\_complete,

name='auth\_password\_reset\_complete'),

url(r'^password\_reset\_done/$', django.contrib.auth.views.password\_reset\_done,

name='auth\_password\_reset\_done'),

url(r'^heartbeat$', include('heartbeat.urls')),

url(r'^user\_api/', include('user\_api.urls')),

url(r'^', include('waffle.urls')),

#xuetangx videoplayer

url(r'^videoid2source/(?P<videoid>.\*?)$', 'video.views.videoid2source', name="videoid2source"),

url(r'^notifystatus$', 'video.views.notify\_status'),

)

#outlink

urlpatterns += (

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/get\_score/(?P<outlink\_id>.\*)', 'outlink.views.get\_score' ,name='get\_score'),

)

# if settings.FEATURES.get("MULTIPLE\_ENROLLMENT\_ROLES"):

urlpatterns += (

url(r'^verify\_student/', include('verify\_student.urls')),

url(r'^course\_modes/', include('course\_modes.urls')),

)

js\_info\_dict = {

'domain': 'djangojs',

# No packages needed, we get LOCALE\_PATHS anyway.

'packages': (),

}

urlpatterns += (

# Serve catalog of localized strings to be rendered by Javascript

url(r'^jsi18n/$', 'django.views.i18n.javascript\_catalog', js\_info\_dict),

)

# sysadmin dashboard, to see what courses are loaded, to delete & load courses

if settings.FEATURES["ENABLE\_SYSADMIN\_DASHBOARD"]:

urlpatterns += (

url(r'^sysadmin/', include('dashboard.sysadmin\_urls')),

)

#Semi-static views (these need to be rendered and have the login bar, but don't change)

urlpatterns += (

url(r'^404$', 'static\_template\_view.views.render',

{'template': '404.html'}, name="404"),

)

# Semi-static views only used by xuetangx, not by themes

if not settings.FEATURES["USE\_CUSTOM\_THEME"]:

urlpatterns += (

url(r'^jobs$', 'static\_template\_view.views.render',

{'template': 'jobs.html'}, name="jobs"),

url(r'^press$', 'student.views.press', name="press"),

url(r'^media-kit$', 'static\_template\_view.views.render',

{'template': 'media-kit.html'}, name="media-kit"),

url(r'^faq$', 'static\_template\_view.views.render',

{'template': 'faq.html'}, name="faq\_ xuetangx "),

url(r'^help$', 'static\_template\_view.views.render',

{'template': 'help.html'}, name="help\_ xuetangx "),

# TODO: (bridger) The copyright has been removed until it is updated for xuetangx

# url(r'^copyright$', 'static\_template\_view.views.render',

# {'template': 'copyright.html'}, name="copyright"),

# Press releases

url(r'^press/([\_a-zA-Z0-9-]+)$', 'static\_template\_view.views.render\_press\_release', name='press\_release'),

# Favicon

(r'^favicon\.ico$', 'django.views.generic.simple.redirect\_to', {'url': '/static/images/favicon.ico'}),

url(r'^submit\_feedback$', 'util.views.submit\_feedback'),

)

# Only enable URLs for those marketing links actually enabled in the

# settings. Disable URLs by marking them as None.

for key, value in settings.MKTG\_URL\_LINK\_MAP.items():

# Skip disabled URLs

if value is None:

continue

# These urls are enabled separately

if key == "ROOT" or key == "COURSES":

continue

# Make the assumptions that the templates are all in the same dir

# and that they all match the name of the key (plus extension)

template = "%s.html" % key.lower()

# To allow theme templates to inherit from default templates,

# prepend a standard prefix

if settings.FEATURES["USE\_CUSTOM\_THEME"]:

template = "theme-" + template

# Make the assumption that the URL we want is the lowercased

# version of the map key

urlpatterns += (url(r'^%s' % key.lower(),

'static\_template\_view.views.render',

{'template': template}, name=value),)

if settings.PERFSTATS:

urlpatterns += (url(r'^reprofile$', 'lms.lib.perfstats.views.end\_profile'),)

# Multicourse wiki (Note: wiki urls must be above the courseware ones because of

# the custom tab catch-all)

if settings.WIKI\_ENABLED:

from wiki.urls import get\_pattern as wiki\_pattern

from django\_notify.urls import get\_pattern as notify\_pattern

# Note that some of these urls are repeated in course\_wiki.course\_nav. Make sure to update

# them together.

urlpatterns += (

# First we include views from course\_wiki that we use to override the default views.

# They come first in the urlpatterns so they get resolved first

url('^wiki/create-root/$', 'course\_wiki.views.root\_create', name='root\_create'),

url(r'^wiki/', include(wiki\_pattern())),

url(r'^notify/', include(notify\_pattern())),

# These urls are for viewing the wiki in the context of a course. They should

# never be returned by a reverse() so they come after the other url patterns

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/course\_wiki/?$',

'course\_wiki.views.course\_wiki\_redirect', name="course\_wiki"),

url(r'^courses/(?:[^/]+/[^/]+/[^/]+)/wiki/', include(wiki\_pattern())),

)

if settings.COURSEWARE\_ENABLED:

urlpatterns += (

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/jump\_to/(?P<location>.\*)$',

'courseware.views.jump\_to', name="jump\_to"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/jump\_to\_id/(?P<module\_id>.\*)$',

'courseware.views.jump\_to\_id', name="jump\_to\_id"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/xblock/(?P<usage\_id>[^/]\*)/handler/(?P<handler>[^/]\*)(?:/(?P<suffix>.\*))?$',

'courseware.module\_render.handle\_xblock\_callback',

name='xblock\_handler'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/xblock/(?P<usage\_id>[^/]\*)/handler\_noauth/(?P<handler>[^/]\*)(?:/(?P<suffix>.\*))?$',

'courseware.module\_render.handle\_xblock\_callback\_noauth',

name='xblock\_handler\_noauth'),

# Software Licenses

# TODO: for now, this is the endpoint of an ajax replay

# service that retrieve and assigns license numbers for

# software assigned to a course. The numbers have to be loaded

# into the database.

url(r'^software-licenses$', 'licenses.views.user\_software\_license', name="user\_software\_license"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/xqueue/(?P<userid>[^/]\*)/(?P<mod\_id>.\*?)/(?P<dispatch>[^/]\*)$',

'courseware.module\_render.xqueue\_callback',

name='xqueue\_callback'),

url(r'^change\_setting$', 'student.views.change\_setting',

name='change\_setting'),

# TODO: These views need to be updated before they work

url(r'^calculate$', 'util.views.calculate'),

# TODO: We should probably remove the circuit package. I believe it was only used in the old way of saving wiki circuits for the wiki

# url(r'^edit\_circuit/(?P<circuit>[^/]\*)$', 'circuit.views.edit\_circuit'),

# url(r'^save\_circuit/(?P<circuit>[^/]\*)$', 'circuit.views.save\_circuit'),

url(r'^courses/search$','courseware.search.search'),

url(r'^courses/?$', 'branding.views.courses', name="courses"),

url(r'^knowledge/(?P<knowledgemap\_id>[^/]+)/$', 'course\_meta.views.knowledgemap\_detail'),

url(r'^partners$', 'course\_meta.views.partners'),

url(r'^change\_enrollment$',

'student.views.change\_enrollment', name="change\_enrollment"),

url(r'^change\_email\_settings$', 'student.views.change\_email\_settings', name="change\_email\_settings"),

#About the course

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/about$',

'courseware.views.course\_about', name="about\_course"),

#View for mktg site (kept for backwards compatibility TODO - remove before merge to master)

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/mktg-about$',

'courseware.views.mktg\_course\_about', name="mktg\_about\_course"),

#View for mktg site

url(r'^mktg/(?P<course\_id>.\*)$',

'courseware.views.mktg\_course\_about', name="mktg\_about\_course"),

#Inside the course

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/$',

'courseware.views.course\_info', name="course\_root"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/info$',

'courseware.views.course\_info', name="info"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/syllabus$',

'courseware.views.syllabus', name="syllabus"), # TODO arjun remove when custom tabs in place, see courseware/courses.py

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/book/(?P<book\_index>\d+)/$',

'staticbook.views.index', name="book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/book/(?P<book\_index>\d+)/(?P<page>\d+)$',

'staticbook.views.index', name="book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/pdfbook/(?P<book\_index>\d+)/$',

'staticbook.views.pdf\_index', name="pdf\_book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/pdfbook/(?P<book\_index>\d+)/(?P<page>\d+)$',

'staticbook.views.pdf\_index', name="pdf\_book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/pdfbook/(?P<book\_index>\d+)/chapter/(?P<chapter>\d+)/$',

'staticbook.views.pdf\_index', name="pdf\_book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/pdfbook/(?P<book\_index>\d+)/chapter/(?P<chapter>\d+)/(?P<page>\d+)$',

'staticbook.views.pdf\_index', name="pdf\_book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/htmlbook/(?P<book\_index>\d+)/$',

'staticbook.views.html\_index', name="html\_book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/htmlbook/(?P<book\_index>\d+)/chapter/(?P<chapter>\d+)/$',

'staticbook.views.html\_index', name="html\_book"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/courseware/?$',

'courseware.views.index', name="courseware"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/courseware/(?P<chapter>[^/]\*)/$',

'courseware.views.index', name="courseware\_chapter"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/courseware/(?P<chapter>[^/]\*)/(?P<section>[^/]\*)/$',

'courseware.views.index', name="courseware\_section"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/courseware/(?P<chapter>[^/]\*)/(?P<section>[^/]\*)/(?P<position>[^/]\*)/?$',

'courseware.views.index', name="courseware\_position"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/progress$',

'courseware.views.progress', name="progress"),

# Takes optional student\_id for instructor use--shows profile as that student sees it.

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/progress/(?P<student\_id>[^/]\*)/$',

'courseware.views.progress', name="student\_progress"),

# For the instructor

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/instructor$',

'instructor.views.legacy.instructor\_dashboard', name="instructor\_dashboard"),

# see ENABLE\_INSTRUCTOR\_BETA\_DASHBOARD section for more urls

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/gradebook$',

'instructor.views.legacy.gradebook', name='gradebook'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/grade\_summary$',

'instructor.views.legacy.grade\_summary', name='grade\_summary'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/staff\_grading$',

'open\_ended\_grading.views.staff\_grading', name='staff\_grading'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/staff\_grading/get\_next$',

'open\_ended\_grading.staff\_grading\_service.get\_next', name='staff\_grading\_get\_next'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/staff\_grading/save\_grade$',

'open\_ended\_grading.staff\_grading\_service.save\_grade', name='staff\_grading\_save\_grade'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/staff\_grading/get\_problem\_list$',

'open\_ended\_grading.staff\_grading\_service.get\_problem\_list', name='staff\_grading\_get\_problem\_list'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/staff\_grading/run\_match$',

'open\_ended\_grading.staff\_grading\_service.run\_match', name='run\_match'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/staff\_grading/clear\_submission$',

'open\_ended\_grading.staff\_grading\_service.clear\_submission', name='clear\_submission'),

# Open Ended problem list

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/open\_ended\_problems$',

'open\_ended\_grading.views.student\_problem\_list', name='open\_ended\_problems'),

# Open Ended flagged problem list

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/open\_ended\_flagged\_problems$',

'open\_ended\_grading.views.flagged\_problem\_list', name='open\_ended\_flagged\_problems'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/open\_ended\_flagged\_problems/take\_action\_on\_flags$',

'open\_ended\_grading.views.take\_action\_on\_flags', name='open\_ended\_flagged\_problems\_take\_action'),

# Cohorts management

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/cohorts$',

'course\_groups.views.list\_cohorts', name="cohorts"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/cohorts/add$',

'course\_groups.views.add\_cohort',

name="add\_cohort"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/cohorts/(?P<cohort\_id>[0-9]+)$',

'course\_groups.views.users\_in\_cohort',

name="list\_cohort"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/cohorts/(?P<cohort\_id>[0-9]+)/add$',

'course\_groups.views.add\_users\_to\_cohort',

name="add\_to\_cohort"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/cohorts/(?P<cohort\_id>[0-9]+)/delete$',

'course\_groups.views.remove\_user\_from\_cohort',

name="remove\_from\_cohort"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/cohorts/debug$',

'course\_groups.views.debug\_cohort\_mgmt',

name="debug\_cohort\_mgmt"),

# Open Ended Notifications

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/open\_ended\_notifications$',

'open\_ended\_grading.views.combined\_notifications', name='open\_ended\_notifications'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/peer\_grading$',

'open\_ended\_grading.views.peer\_grading', name='peer\_grading'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/notes$', 'notes.views.notes', name='notes'),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/notes/', include('notes.urls')),

# mobile index page

url(r'^mobile/?$', 'mobile.views.index', name='mobile'),

)

# allow course staff to change to student view of courseware

if settings.FEATURES.get('ENABLE\_MASQUERADE'):

urlpatterns += (

url(r'^masquerade/(?P<marg>.\*)$', 'courseware.masquerade.handle\_ajax', name="masquerade-switch"),

)

# discussion forums live within courseware, so courseware must be enabled first

if settings.FEATURES.get('ENABLE\_DISCUSSION\_SERVICE'):

urlpatterns += (

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/discussion/',

include('django\_comment\_client.urls')),

url(r'^notification\_prefs/enable/', 'notification\_prefs.views.ajax\_enable'),

url(r'^notification\_prefs/disable/', 'notification\_prefs.views.ajax\_disable'),

url(r'^notification\_prefs/status/', 'notification\_prefs.views.ajax\_status'),

url(r'^notification\_prefs/unsubscribe/(?P<token>[a-zA-Z0-9-\_=]+)/',

'notification\_prefs.views.set\_subscription', {'subscribe': False}, name="unsubscribe\_forum\_update"),

url(r'^notification\_prefs/resubscribe/(?P<token>[a-zA-Z0-9-\_=]+)/',

'notification\_prefs.views.set\_subscription', {'subscribe': True}, name="resubscribe\_forum\_update"),

)

urlpatterns += (

# This MUST be the last view in the courseware--it's a catch-all for custom tabs.

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/(?P<tab\_slug>[^/]+)/$',

'courseware.views.static\_tab', name="static\_tab"),

)

if settings.FEATURES.get('ENABLE\_STUDENT\_HISTORY\_VIEW'):

urlpatterns += (

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/submission\_history/(?P<student\_username>[^/]\*)/(?P<location>.\*?)$',

'courseware.views.submission\_history',

name='submission\_history'),

)

if settings.COURSEWARE\_ENABLED and settings.FEATURES.get('ENABLE\_INSTRUCTOR\_BETA\_DASHBOARD'):

urlpatterns += (

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/instructor\_dashboard$',

'instructor.views.instructor\_dashboard.instructor\_dashboard\_2', name="instructor\_dashboard\_2"),

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/instructor\_dashboard/api/',

include('instructor.views.api\_urls'))

)

if settings.DEBUG or settings.FEATURES.get('ENABLE\_DJANGO\_ADMIN\_SITE'):

## Jasmine and admin

urlpatterns += (url(r'^admin/', include(admin.site.urls)),)

if settings.FEATURES.get('AUTH\_USE\_OPENID'):

urlpatterns += (

url(r'^openid/login/$', 'django\_openid\_auth.views.login\_begin', name='openid-login'),

url(r'^openid/complete/$', 'external\_auth.views.openid\_login\_complete', name='openid-complete'),

url(r'^openid/logo.gif$', 'django\_openid\_auth.views.logo', name='openid-logo'),

)

if settings.FEATURES.get('AUTH\_USE\_SHIB'):

urlpatterns += (

url(r'^shib-login/$', 'external\_auth.views.shib\_login', name='shib-login'),

)

if settings.FEATURES.get('AUTH\_USE\_CAS'):

urlpatterns += (

url(r'^cas-auth/login/$', 'external\_auth.views.cas\_login', name="cas-login"),

url(r'^cas-auth/logout/$', 'django\_cas.views.logout', {'next\_page': '/'}, name="cas-logout"),

)

if settings.FEATURES.get('RESTRICT\_ENROLL\_BY\_REG\_METHOD'):

urlpatterns += (

url(r'^course\_specific\_login/(?P<course\_id>[^/]+/[^/]+/[^/]+)/$',

'external\_auth.views.course\_specific\_login', name='course-specific-login'),

url(r'^course\_specific\_register/(?P<course\_id>[^/]+/[^/]+/[^/]+)/$',

'external\_auth.views.course\_specific\_register', name='course-specific-register'),

)

# Shopping cart

urlpatterns += (

url(r'^shoppingcart/', include('shoppingcart.urls')),

)

if settings.FEATURES.get('AUTH\_USE\_OPENID\_PROVIDER'):

urlpatterns += (

url(r'^openid/provider/login/$', 'external\_auth.views.provider\_login', name='openid-provider-login'),

url(r'^openid/provider/login/(?:.+)$', 'external\_auth.views.provider\_identity', name='openid-provider-login-identity'),

url(r'^openid/provider/identity/$', 'external\_auth.views.provider\_identity', name='openid-provider-identity'),

url(r'^openid/provider/xrds/$', 'external\_auth.views.provider\_xrds', name='openid-provider-xrds')

)

if settings.FEATURES.get('ENABLE\_LMS\_MIGRATION'):

urlpatterns += (

url(r'^migrate/modules$', 'lms\_migration.migrate.manage\_modulestores'),

url(r'^migrate/reload/(?P<reload\_dir>[^/]+)$', 'lms\_migration.migrate.manage\_modulestores'),

url(r'^migrate/reload/(?P<reload\_dir>[^/]+)/(?P<commit\_id>[^/]+)$', 'lms\_migration.migrate.manage\_modulestores'),

url(r'^gitreload$', 'lms\_migration.migrate.gitreload'),

url(r'^gitreload/(?P<reload\_dir>[^/]+)$', 'lms\_migration.migrate.gitreload'),

)

if settings.FEATURES.get('ENABLE\_SQL\_TRACKING\_LOGS'):

urlpatterns += (

url(r'^event\_logs$', 'track.views.view\_tracking\_log'),

url(r'^event\_logs/(?P<args>.+)$', 'track.views.view\_tracking\_log'),

)

if settings.FEATURES.get('ENABLE\_SERVICE\_STATUS'):

urlpatterns += (

url(r'^status/', include('service\_status.urls')),

)

if settings.FEATURES.get('ENABLE\_INSTRUCTOR\_BACKGROUND\_TASKS'):

urlpatterns += (

url(r'^instructor\_task\_status/$', 'instructor\_task.views.instructor\_task\_status', name='instructor\_task\_status'),

)

if settings.FEATURES.get('RUN\_AS\_ANALYTICS\_SERVER\_ENABLED'):

urlpatterns += (

url(r'^edinsights\_service/', include('edinsights.core.urls')),

)

import edinsights.core.registry

# FoldIt views

urlpatterns += (

# The path is hardcoded into their app...

url(r'^comm/foldit\_ops', 'foldit.views.foldit\_ops', name="foldit\_ops"),

)

if settings.FEATURES.get('ENABLE\_DEBUG\_RUN\_PYTHON'):

urlpatterns += (

url(r'^debug/run\_python', 'debug.views.run\_python'),

)

# Crowdsourced hinting instructor manager.

if settings.FEATURES.get('ENABLE\_HINTER\_INSTRUCTOR\_VIEW'):

urlpatterns += (

url(r'^courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)/hint\_manager$',

'instructor.hint\_manager.hint\_manager', name="hint\_manager"),

)

# enable automatic login

if settings.FEATURES.get('AUTOMATIC\_AUTH\_FOR\_TESTING'):

urlpatterns += (

url(r'^auto\_auth$', 'student.views.auto\_auth'),

)

urlpatterns += (

url(r'^notsupport$', 'static\_template\_view.views.render',

{'template': 'notsupport.html'}, name="notsupport"),

)

urlpatterns = patterns(\*urlpatterns)

if settings.DEBUG:

urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

#Custom error pages

handler404 = 'static\_template\_view.views.render\_404'

handler500 = 'static\_template\_view.views.render\_500'

## 教师工具部分

**Cms/urls.py**

from django.conf import settings

from django.conf.urls import patterns, include, url

from xmodule.modulestore import parsers

# There is a course creators admin table.

from ratelimitbackend import admin

admin.autodiscover()

urlpatterns = patterns('', # nopep8

url(r'^transcripts/upload$', 'contentstore.views.upload\_transcripts', name='upload\_transcripts'),

url(r'^transcripts/download$', 'contentstore.views.download\_transcripts', name='download\_transcripts'),

url(r'^transcripts/check$', 'contentstore.views.check\_transcripts', name='check\_transcripts'),

url(r'^transcripts/choose$', 'contentstore.views.choose\_transcripts', name='choose\_transcripts'),

url(r'^transcripts/replace$', 'contentstore.views.replace\_transcripts', name='replace\_transcripts'),

url(r'^transcripts/rename$', 'contentstore.views.rename\_transcripts', name='rename\_transcripts'),

url(r'^transcripts/save$', 'contentstore.views.save\_transcripts', name='save\_transcripts'),

url(r'^preview/xblock/(?P<usage\_id>.\*?)/handler/(?P<handler>[^/]\*)(?:/(?P<suffix>.\*))?$',

'contentstore.views.preview\_handler', name='preview\_handler'),

url(r'^xblock/(?P<usage\_id>.\*?)/handler/(?P<handler>[^/]\*)(?:/(?P<suffix>.\*))?$',

'contentstore.views.component\_handler', name='component\_handler'),

# temporary landing page for a course

url(r'^edge/(?P<org>[^/]+)/(?P<course>[^/]+)/course/(?P<coursename>[^/]+)$',

'contentstore.views.landing', name='landing'),

url(r'^not\_found$', 'contentstore.views.not\_found', name='not\_found'),

url(r'^server\_error$', 'contentstore.views.server\_error', name='server\_error'),

# temporary landing page for edge

url(r'^edge$', 'contentstore.views.edge', name='edge'),

# noop to squelch ajax errors

url(r'^event$', 'contentstore.views.event', name='event'),

url(r'^xmodule/', include('pipeline\_js.urls')),

url(r'^heartbeat$', include('heartbeat.urls')),

#xuetangx videoplayer

url(r'^videoid2source/(?P<videoid>.\*?)$', 'contentstore.views.videoid2source', name='videoid2source'),

url(r'^checkpermission/(?P<querystring>.\*?)$', 'contentstore.views.check\_permission', name='check\_permission'),

url(r'^notifystatus$', 'contentstore.views.notify\_status', name='notify\_status'),

url(r'^course\_list$', 'course\_meta.views.course\_list'),

)

#outlink

urlpatterns += patterns(

'',

url(r'^courses/none/get\_score/(?P<outlink\_id>.\*)','contentstore.views.get\_score' ,name='getScore'),

)

# User creation and updating views

urlpatterns += patterns(

'',

url(r'^create\_account$', 'student.views.create\_account', name='create\_account'),

url(r'^activate/(?P<key>[^/]\*)$', 'student.views.activate\_account', name='activate'),

# ajax view that actually does the work

url(r'^login\_post$', 'student.views.login\_user', name='login\_post'),

url(r'^logout$', 'student.views.logout\_user', name='logout'),

)

# restful api

urlpatterns += patterns(

'contentstore.views',

url(r'^$', 'howitworks', name='homepage'),

url(r'^howitworks$', 'howitworks'),

url(r'^signup$', 'signup', name='signup'),

url(r'^signin$', 'login\_page', name='login'),

url(r'^request\_course\_creator$', 'request\_course\_creator'),

# (?ix) == ignore case and verbose (multiline regex)

url(r'(?ix)^course\_team/{}(/)?(?P<email>.+)?$'.format(parsers.URL\_RE\_SOURCE), 'course\_team\_handler'),

url(r'(?ix)^course\_info/{}$'.format(parsers.URL\_RE\_SOURCE), 'course\_info\_handler'),

url(

r'(?ix)^course\_info\_update/{}(/)?(?P<provided\_id>\d+)?$'.format(parsers.URL\_RE\_SOURCE),

'course\_info\_update\_handler'

),

url(r'(?ix)^course($|/){}$'.format(parsers.URL\_RE\_SOURCE), 'course\_handler'),

url(r'(?ix)^subsection($|/){}$'.format(parsers.URL\_RE\_SOURCE), 'subsection\_handler'),

url(r'(?ix)^unit($|/){}$'.format(parsers.URL\_RE\_SOURCE), 'unit\_handler'),

url(r'(?ix)^checklists/{}(/)?(?P<checklist\_index>\d+)?$'.format(parsers.URL\_RE\_SOURCE), 'checklists\_handler'),

url(r'(?ix)^orphan/{}$'.format(parsers.URL\_RE\_SOURCE), 'orphan\_handler'),

url(r'(?ix)^assets/{}(/)?(?P<asset\_id>.+)?$'.format(parsers.URL\_RE\_SOURCE), 'assets\_handler'),

url(r'(?ix)^import/{}$'.format(parsers.URL\_RE\_SOURCE), 'import\_handler'),

url(r'(?ix)^import\_status/{}/(?P<filename>.+)$'.format(parsers.URL\_RE\_SOURCE), 'import\_status\_handler'),

url(r'(?ix)^export/{}$'.format(parsers.URL\_RE\_SOURCE), 'export\_handler'),

url(r'(?ix)^xblock($|/){}$'.format(parsers.URL\_RE\_SOURCE), 'xblock\_handler'),

url(r'(?ix)^tabs/{}$'.format(parsers.URL\_RE\_SOURCE), 'tabs\_handler'),

url(r'(?ix)^settings/details/{}$'.format(parsers.URL\_RE\_SOURCE), 'settings\_handler'),

url(r'(?ix)^settings/grading/{}(/)?(?P<grader\_index>\d+)?$'.format(parsers.URL\_RE\_SOURCE), 'grading\_handler'),

url(r'(?ix)^settings/advanced/{}$'.format(parsers.URL\_RE\_SOURCE), 'advanced\_settings\_handler'),

url(r'(?ix)^textbooks/{}$'.format(parsers.URL\_RE\_SOURCE), 'textbooks\_list\_handler'),

url(r'(?ix)^textbooks/{}/(?P<tid>\d[^/]\*)$'.format(parsers.URL\_RE\_SOURCE), 'textbooks\_detail\_handler'),

)

js\_info\_dict = {

'domain': 'djangojs',

# No packages needed, we get LOCALE\_PATHS anyway.

'packages': (),

}

urlpatterns += patterns('',

# Serve catalog of localized strings to be rendered by Javascript

url(r'^i18n.js$', 'django.views.i18n.javascript\_catalog', js\_info\_dict),

)

if settings.FEATURES.get('ENABLE\_SERVICE\_STATUS'):

urlpatterns += patterns('',

url(r'^status/', include('service\_status.urls')),

)

urlpatterns += patterns('', url(r'^admin/', include(admin.site.urls)),)

# enable automatic login

if settings.FEATURES.get('AUTOMATIC\_AUTH\_FOR\_TESTING'):

urlpatterns += (

url(r'^auto\_auth$', 'student.views.auto\_auth'),

)

if settings.DEBUG:

try:

from .urls\_dev import urlpatterns as dev\_urlpatterns

urlpatterns += dev\_urlpatterns

except ImportError:

pass

# Custom error pages

#pylint: disable=C0103

handler404 = 'contentstore.views.render\_404'

handler500 = 'contentstore.views.render\_500'

## 学生功能实现

**Common/djangoapps/student/views.py**

"""

Student Views

"""

import datetime

import json

import logging

import re

import urllib

import uuid

import time

from collections import defaultdict

from pytz import UTC

from django.conf import settings

from django.contrib.auth import logout, authenticate, login

from django.contrib.auth.models import User, AnonymousUser

from django.contrib.auth.decorators import login\_required

from django.contrib.auth.views import password\_reset\_confirm

from django.core.cache import cache

from django.core.context\_processors import csrf

from django.core.mail import send\_mail

from django.core.urlresolvers import reverse

from django.core.validators import validate\_email, validate\_slug, ValidationError

from django.db import IntegrityError, transaction

from django.http import (HttpResponse, HttpResponseBadRequest, HttpResponseForbidden,

Http404)

from django.shortcuts import redirect

from django\_future.csrf import ensure\_csrf\_cookie, csrf\_exempt

from django.utils.http import cookie\_date, base36\_to\_int

from django.utils.translation import ugettext as \_

from django.views.decorators.http import require\_POST, require\_GET

from ratelimitbackend.exceptions import RateLimitException

from xuetangxmako.shortcuts import render\_to\_response, render\_to\_string

from course\_meta.models import Course, HomepageCourses

from course\_meta.utils import naturalFormatDate, queryDataService

from course\_modes.models import CourseMode

from student.models import (

Registration, UserProfile, PendingNameChange,

PendingEmailChange, CourseEnrollment, unique\_id\_for\_user,

CourseEnrollmentAllowed, UserStanding, LoginFailures

)

from student.forms import PasswordResetFormNoActive

from student.firebase\_token\_generator import create\_token

from course\_meta.utils import naturalFormatDate

from verify\_student.models import SoftwareSecurePhotoVerification, MidcourseReverificationWindow

from certificates.models import CertificateStatuses, certificate\_status\_for\_student

from xmodule.course\_module import CourseDescriptor

from xmodule.modulestore.exceptions import ItemNotFoundError

from xmodule.modulestore.django import modulestore

from xmodule.modulestore import MONGO\_MODULESTORE\_TYPE

from collections import namedtuple

from courseware.courses import get\_courses, sort\_by\_announcement, get\_course\_with\_access

from courseware.access import has\_access

from external\_auth.models import ExternalAuthMap

import external\_auth.views

from bulk\_email.models import Optout, CourseAuthorization

import shoppingcart

import track.views

from dogapi import dog\_stats\_api

from util.json\_request import JsonResponse

from microsite\_configuration.middleware import MicrositeConfiguration

from util.password\_policy\_validators import (

validate\_password\_length, validate\_password\_complexity,

validate\_password\_dictionary

)

from courseware.courses import get\_course\_info\_section

from lxml import html

from courseware.courses import get\_course\_info\_section

from lxml import html

log = logging.getLogger("xuetangx.student")

AUDIT\_LOG = logging.getLogger("audit")

Article = namedtuple('Article', 'title url author image deck publication publish\_date')

ReverifyInfo = namedtuple('ReverifyInfo', 'course\_id course\_name course\_number date status display') # pylint: disable=C0103

def csrf\_token(context):

"""A csrf token that can be included in a form."""

csrf\_token = context.get('csrf\_token', '')

if csrf\_token == 'NOTPROVIDED':

return ''

return (u'<div style="display:none"><input type="hidden"'

' name="csrfmiddlewaretoken" value="%s" /></div>' % (csrf\_token))

# NOTE: This view is not linked to directly--it is called from

# branding/views.py:index(), which is cached for anonymous users.

# This means that it should always return the same thing for anon

# users. (in particular, no switching based on query params allowed)

def index(request, extra\_context={}, user=AnonymousUser()):

# Get 8 hot courses in homepage

homepage\_course\_list = HomepageCourses.objects.order\_by('order')[0:8]

hot\_courses = []

for homepage\_course in homepage\_course\_list:

course = homepage\_course.course

course\_info = \_get\_course\_info(course)

hot\_courses.append(course\_info)

# Get 8 recent courses

now = datetime.datetime.now(UTC)

courses = Course.objects.filter(start\_\_gt=now).order\_by('start')[0:8]

recent\_courses = []

for course in courses:

course\_info = \_get\_course\_info(course)

if course.start:

course\_info['start'] = naturalFormatDate(course.start)

recent\_courses.append(course\_info)

# If less than 8 courses, get the recent started courses

if len(recent\_courses) < 8:

courses = Course.objects.filter(start\_\_lte=now).order\_by('-start')[0:8-len(recent\_courses)]

for course in courses:

course\_info = \_get\_course\_info(course)

if course.start:

course\_info['start'] = naturalFormatDate(course.start)

recent\_courses.append(course\_info)

# Get the 8 newest courses

courses = Course.objects.filter(status\_\_gte=0).filter(owner='xuetangX').order\_by('-create\_time')[0:8]

new\_courses = []

for course in courses:

course\_info = \_get\_course\_info(course)

new\_courses.append(course\_info)

# Count of all courses

course\_count = Course.objects.count()

return render\_to\_response('xuetangx/index.html', {

'hot\_courses': hot\_courses,

'recent\_courses': recent\_courses,

'new\_courses': new\_courses,

'course\_count': course\_count,

})

def \_get\_course\_info(course):

course\_info = {}

course\_info['course\_id'] = course.course\_id

course\_info['name'] = course.name

course\_info['thumbnail'] = course.thumbnail

course\_info['course\_num'] = course.course\_num

course\_info['modified'] = naturalFormatDate(course.modified)

course\_info['enrollment'] = queryDataService(

course.course\_id, 'ENROLL\_TOTAL')

course\_info['comment'] = queryDataService(course.course\_id, 'COMMENTS')

course\_info['subtitle'] = course.subtitle

try:

staff = course.staff.all()[0]

course\_info['staff\_avatar'] = staff.avartar

course\_info['staff\_name'] = staff.name

course\_info['staff\_title'] = "%s %s %s" % (

staff.company, staff.department, staff.position)

except:

course\_info['staff\_avatar'] = ''

course\_info['staff\_name'] = ''

course\_info['staff\_title'] = ''

log.warning(

u"Course {0} is missing staff information.".format(course.course\_id))

return course\_info

def course\_from\_id(course\_id):

"""Return the CourseDescriptor corresponding to this course\_id"""

course\_loc = CourseDescriptor.id\_to\_location(course\_id)

return modulestore().get\_instance(course\_id, course\_loc)

day\_pattern = re.compile(r'\s\d+,\s')

multimonth\_pattern = re.compile(r'\s?\-\s?\S+\s')

def \_get\_date\_for\_press(publish\_date):

# strip off extra months, and just use the first:

date = re.sub(multimonth\_pattern, ", ", publish\_date)

if re.search(day\_pattern, date):

date = datetime.datetime.strptime(date, "%B %d, %Y").replace(tzinfo=UTC)

else:

date = datetime.datetime.strptime(date, "%B, %Y").replace(tzinfo=UTC)

return date

def press(request):

json\_articles = cache.get("student\_press\_json\_articles")

if json\_articles is None:

if hasattr(settings, 'RSS\_URL'):

content = urllib.urlopen(settings.PRESS\_URL).read()

json\_articles = json.loads(content)

else:

content = open(settings.PROJECT\_ROOT /

"templates" / "press.json").read()

json\_articles = json.loads(content)

cache.set("student\_press\_json\_articles", json\_articles)

articles = [Article(\*\*article) for article in json\_articles]

articles.sort(key=lambda item: \_get\_date\_for\_press(item.publish\_date), reverse=True)

return render\_to\_response('static\_templates/press.html', {'articles': articles})

def process\_survey\_link(survey\_link, user):

"""

If {UNIQUE\_ID} appears in the link, replace it with a unique id for the user.

Currently, this is sha1(user.username). Otherwise, return survey\_link.

"""

return survey\_link.format(UNIQUE\_ID=unique\_id\_for\_user(user))

def cert\_info(user, course):

"""

Get the certificate info needed to render the dashboard section for the given

student and course. Returns a dictionary with keys:

'status': one of 'generating', 'ready', 'notpassing', 'processing', 'restricted'

'show\_download\_url': bool

'download\_url': url, only present if show\_download\_url is True

'show\_disabled\_download\_button': bool -- true if state is 'generating'

'show\_survey\_button': bool

'survey\_url': url, only if show\_survey\_button is True

'grade': if status is not 'processing'

"""

if not course.has\_ended():

return {}

return \_cert\_info(user, course, certificate\_status\_for\_student(user, course.id))

def reverification\_info(course\_enrollment\_pairs, user, statuses):

"""

Returns reverification-related information for \*all\* of user's enrollments whose

reverification status is in status\_list

Args:

course\_enrollment\_pairs (list): list of (course, enrollment) tuples

user (User): the user whose information we want

statuses (list): a list of reverification statuses we want information for

example: ["must\_reverify", "denied"]

Returns:

dictionary of lists: dictionary with one key per status, e.g.

dict["must\_reverify"] = []

dict["must\_reverify"] = [some information]

"""

reverifications = defaultdict(list)

for (course, enrollment) in course\_enrollment\_pairs:

info = single\_course\_reverification\_info(user, course, enrollment)

if info:

reverifications[info.status].append(info)

# Sort the data by the reverification\_end\_date

for status in statuses:

if reverifications[status]:

reverifications[status].sort(key=lambda x: x.date)

return reverifications

def single\_course\_reverification\_info(user, course, enrollment): # pylint: disable=invalid-name

"""Returns midcourse reverification-related information for user with enrollment in course.

If a course has an open re-verification window, and that user has a verified enrollment in

the course, we return a tuple with relevant information. Returns None if there is no info..

Args:

user (User): the user we want to get information for

course (Course): the course in which the student is enrolled

enrollment (CourseEnrollment): the object representing the type of enrollment user has in course

Returns:

ReverifyInfo: (course\_id, course\_name, course\_number, date, status)

OR, None: None if there is no re-verification info for this enrollment

"""

window = MidcourseReverificationWindow.get\_window(course.id, datetime.datetime.now(UTC))

# If there's no window OR the user is not verified, we don't get reverification info

if (not window) or (enrollment.mode != "verified"):

return None

return ReverifyInfo(

course.id, course.display\_name, course.number,

window.end\_date.strftime('%B %d, %Y %X %p'),

SoftwareSecurePhotoVerification.user\_status(user, window)[0],

SoftwareSecurePhotoVerification.display\_status(user, window),

)

def get\_course\_enrollment\_pairs(user, course\_org\_filter, org\_filter\_out\_set):

"""

Get the relevant set of (Course, CourseEnrollment) pairs to be displayed on

a student's dashboard.

"""

for enrollment in CourseEnrollment.enrollments\_for\_user(user):

try:

course = course\_from\_id(enrollment.course\_id)

# if we are in a Microsite, then filter out anything that is not

# attributed (by ORG) to that Microsite

if course\_org\_filter and course\_org\_filter != course.location.org:

continue

# Conversely, if we are not in a Microsite, then let's filter out any enrollments

# with courses attributed (by ORG) to Microsites

elif course.location.org in org\_filter\_out\_set:

continue

yield (course, enrollment)

except ItemNotFoundError:

log.error("User {0} enrolled in non-existent course {1}"

.format(user.username, enrollment.course\_id))

def \_cert\_info(user, course, cert\_status):

"""

Implements the logic for cert\_info -- split out for testing.

"""

default\_status = 'processing'

default\_info = {'status': default\_status,

'show\_disabled\_download\_button': False,

'show\_download\_url': False,

'show\_survey\_button': False,

}

if cert\_status is None:

return default\_info

# simplify the status for the template using this lookup table

template\_state = {

CertificateStatuses.generating: 'generating',

CertificateStatuses.regenerating: 'generating',

CertificateStatuses.downloadable: 'ready',

CertificateStatuses.notpassing: 'notpassing',

CertificateStatuses.restricted: 'restricted',

}

status = template\_state.get(cert\_status['status'], default\_status)

d = {'status': status,

'show\_download\_url': status == 'ready',

'show\_disabled\_download\_button': status == 'generating',

'mode': cert\_status.get('mode', None)}

if (status in ('generating', 'ready', 'notpassing', 'restricted') and

course.end\_of\_course\_survey\_url is not None):

d.update({

'show\_survey\_button': True,

'survey\_url': process\_survey\_link(course.end\_of\_course\_survey\_url, user)})

else:

d['show\_survey\_button'] = False

if status == 'ready':

if 'download\_url' not in cert\_status:

log.warning(

"User %s has a downloadable cert for %s, but no download url",

user.username, course.id)

return default\_info

else:

d['download\_url'] = cert\_status['download\_url']

if status in ('generating', 'ready', 'notpassing', 'restricted'):

if 'grade' not in cert\_status:

# Note: as of 11/20/2012, we know there are students in this state-- cs169.1x,

# who need to be regraded (we weren't tracking 'notpassing' at first).

# We can add a log.warning here once we think it shouldn't happen.

return default\_info

else:

d['grade'] = cert\_status['grade']

return d

@ensure\_csrf\_cookie

def signin\_user(request):

"""

This view will display the non-modal login form

"""

if (settings.FEATURES['AUTH\_USE\_CERTIFICATES'] and

external\_auth.views.ssl\_get\_cert\_from\_request(request)):

# SSL login doesn't require a view, so redirect

# branding and allow that to process the login if it

# is enabled and the header is in the request.

return redirect(reverse('root'))

if request.user.is\_authenticated():

return redirect(reverse('dashboard'))

context = {

'course\_id': request.GET.get('course\_id'),

'enrollment\_action': request.GET.get('enrollment\_action'),

'platform\_name': MicrositeConfiguration.get\_microsite\_configuration\_value(

'platform\_name',

settings.PLATFORM\_NAME

),

}

return render\_to\_response('login.html', context)

@ensure\_csrf\_cookie

def register\_user(request, extra\_context=None):

"""

This view will display the non-modal registration form

"""

if request.user.is\_authenticated():

return redirect(reverse('dashboard'))

if settings.FEATURES.get('AUTH\_USE\_CERTIFICATES\_IMMEDIATE\_SIGNUP'):

# Redirect to branding to process their certificate if SSL is enabled

# and registration is disabled.

return redirect(reverse('root'))

context = {

'course\_id': request.GET.get('course\_id'),

'enrollment\_action': request.GET.get('enrollment\_action'),

'platform\_name': MicrositeConfiguration.get\_microsite\_configuration\_value(

'platform\_name',

settings.PLATFORM\_NAME

),

}

if extra\_context is not None:

context.update(extra\_context)

if context.get("extauth\_domain", '').startswith(external\_auth.views.SHIBBOLETH\_DOMAIN\_PREFIX):

return render\_to\_response('register-shib.html', context)

return render\_to\_response('register.html', context)

def complete\_course\_mode\_info(course\_id, enrollment):

"""

We would like to compute some more information from the given course modes

and the user's current enrollment

Returns the given information:

- whether to show the course upsell information

- numbers of days until they can't upsell anymore

"""

modes = CourseMode.modes\_for\_course\_dict(course\_id)

mode\_info = {'show\_upsell': False, 'days\_for\_upsell': None}

# we want to know if the user is already verified and if verified is an

# option

if 'verified' in modes and enrollment.mode != 'verified':

mode\_info['show\_upsell'] = True

# if there is an expiration date, find out how long from now it is

if modes['verified'].expiration\_datetime:

today = datetime.datetime.now(UTC).date()

mode\_info['days\_for\_upsell'] = (modes['verified'].expiration\_datetime.date() - today).days

return mode\_info

@login\_required

@ensure\_csrf\_cookie

def dashboard(request):

user = request.user

# for microsites, we want to filter and only show enrollments for courses within

# the microsites 'ORG'

course\_org\_filter = MicrositeConfiguration.get\_microsite\_configuration\_value('course\_org\_filter')

# Let's filter out any courses in an "org" that has been declared to be

# in a Microsite

org\_filter\_out\_set = MicrositeConfiguration.get\_all\_microsite\_orgs()

# remove our current Microsite from the "filter out" list, if applicable

if course\_org\_filter:

org\_filter\_out\_set.remove(course\_org\_filter)

# Build our (course, enrollment) list for the user, but ignore any courses that no

# longer exist (because the course IDs have changed). Still, we don't delete those

# enrollments, because it could have been a data push snafu.

course\_enrollment\_pairs = list(get\_course\_enrollment\_pairs(user, course\_org\_filter, org\_filter\_out\_set))

course\_optouts = Optout.objects.filter(user=user).values\_list('course\_id', flat=True)

message = ""

if not user.is\_active:

message = render\_to\_string(

'registration/activate\_account\_notice.html', {'email': user.email})

# Global staff can see what courses errored on their dashboard

staff\_access = False

errored\_courses = {}

if has\_access(user, 'global', 'staff'):

# Show any courses that errored on load

staff\_access = True

errored\_courses = modulestore().get\_errored\_courses()

show\_courseware\_links\_for = frozenset(course.id for course, \_enrollment in course\_enrollment\_pairs

if has\_access(request.user, course, 'load'))

course\_modes = {course.id: complete\_course\_mode\_info(course.id, enrollment) for course, enrollment in course\_enrollment\_pairs}

cert\_statuses = {course.id: cert\_info(request.user, course) for course, \_enrollment in course\_enrollment\_pairs}

# only show email settings for Mongo course and when bulk email is turned on

show\_email\_settings\_for = frozenset(

course.id for course, \_enrollment in course\_enrollment\_pairs if (

settings.FEATURES['ENABLE\_INSTRUCTOR\_EMAIL'] and

modulestore().get\_modulestore\_type(course.id) == MONGO\_MODULESTORE\_TYPE and

CourseAuthorization.instructor\_email\_enabled(course.id)

)

)

# Verification Attempts

# Used to generate the "you must reverify for course x" banner

verification\_status, verification\_msg = SoftwareSecurePhotoVerification.user\_status(user)

# Gets data for midcourse reverifications, if any are necessary or have failed

statuses = ["approved", "denied", "pending", "must\_reverify"]

reverifications = reverification\_info(course\_enrollment\_pairs, user, statuses)

show\_refund\_option\_for = frozenset(course.id for course, \_enrollment in course\_enrollment\_pairs

if \_enrollment.refundable())

# get info w.r.t ExternalAuthMap

external\_auth\_map = None

try:

external\_auth\_map = ExternalAuthMap.objects.get(user=user)

except ExternalAuthMap.DoesNotExist:

pass

course\_updates = {}

for course\_tmp,enrollment in course\_enrollment\_pairs:

try:

course = get\_course\_with\_access(request.user, course\_tmp.id, 'load')

updates = get\_course\_info\_section(request, course, 'updates')

if len(updates) <= 1:

continue

first\_update = get\_first\_update(updates)

if not first\_update:

continue

h3\_match = re.match(r"<h\d>.\*?</h\d>",first\_update['content'])

if h3\_match:

update\_con = h3\_match.group(0)

update\_con = re.sub(r"<h\d>","",update\_con)

update\_con = re.sub(r"</h\d>","",update\_con)

else:

update\_con = re.sub(r"<.\*?>","",first\_update['content'])

if len(update\_con) > 290:

update\_con = update\_con[0:290]

if '&' in update\_con[284:290]:

update\_con = update\_con[0:update\_con[284:290].find('&') + 284]

update\_con += "..."

course\_updates[course\_tmp.id] = {'date':first\_update['date'], 'content':update\_con}

except:

pass

# If there are \*any\* denied reverifications that have not been toggled off,

# we'll display the banner

denied\_banner = any(item.display for item in reverifications["denied"])

context = {'course\_enrollment\_pairs': course\_enrollment\_pairs,

'course\_optouts': course\_optouts,

'message': message,

'external\_auth\_map': external\_auth\_map,

'staff\_access': staff\_access,

'errored\_courses': errored\_courses,

'show\_courseware\_links\_for': show\_courseware\_links\_for,

'all\_course\_modes': course\_modes,

'cert\_statuses': cert\_statuses,

'show\_email\_settings\_for': show\_email\_settings\_for,

'reverifications': reverifications,

'verification\_status': verification\_status,

'verification\_msg': verification\_msg,

'show\_refund\_option\_for': show\_refund\_option\_for,

'denied\_banner': denied\_banner,

'billing\_email': settings.PAYMENT\_SUPPORT\_EMAIL,

'email': request.user.email,

'user\_id': str(request.user.id),

'updates': course\_updates,

}

return render\_to\_response('dashboard.html', context)

def get\_first\_update(update\_string):

"""

TODO: cannot parse content which has a '<li></li>' tag

"""

update\_string = update\_string.strip()

log\_pattern = re.compile(r'<ol><li><h2>(.\*?)</h2>(.\*?)</li>.\*</ol>', flags=re.DOTALL)

match = log\_pattern.match(update\_string)

if match:

return {

'date': match.group(1),

'content': match.group(2)

}

else:

return None

def try\_change\_enrollment(request):

"""

This method calls change\_enrollment if the necessary POST

parameters are present, but does not return anything. It

simply logs the result or exception. This is usually

called after a registration or login, as secondary action.

It should not interrupt a successful registration or login.

"""

if 'enrollment\_action' in request.POST:

try:

enrollment\_response = change\_enrollment(request)

# There isn't really a way to display the results to the user, so we just log it

# We expect the enrollment to be a success, and will show up on the

# dashboard anyway

log.info(

"Attempted to automatically enroll after login. Response code: {0}; response body: {1}".format(

enrollment\_response.status\_code,

enrollment\_response.content

)

)

if enrollment\_response.content != '':

return enrollment\_response.content

except Exception, e:

log.exception(

"Exception automatically enrolling after login: {0}".format(str(e)))

@require\_POST

def change\_enrollment(request):

"""

Modify the enrollment status for the logged-in user.

The request parameter must be a POST request (other methods return 405)

that specifies course\_id and enrollment\_action parameters. If course\_id or

enrollment\_action is not specified, if course\_id is not valid, if

enrollment\_action is something other than "enroll" or "unenroll", if

enrollment\_action is "enroll" and enrollment is closed for the course, or

if enrollment\_action is "unenroll" and the user is not enrolled in the

course, a 400 error will be returned. If the user is not logged in, 403

will be returned; it is important that only this case return 403 so the

front end can redirect the user to a registration or login page when this

happens. This function should only be called from an AJAX request or

as a post-login/registration helper, so the error messages in the responses

should never actually be user-visible.

"""

user = request.user

action = request.POST.get("enrollment\_action")

course\_id = request.POST.get("course\_id")

if course\_id is None:

return HttpResponseBadRequest(\_("Course id not specified"))

if not user.is\_authenticated():

return HttpResponseForbidden()

if action == "enroll":

# Make sure the course exists

# We don't do this check on unenroll, or a bad course id can't be

# unenrolled from

try:

course = course\_from\_id(course\_id)

except ItemNotFoundError:

log.warning("User {0} tried to enroll in non-existent course {1}"

.format(user.username, course\_id))

return HttpResponseBadRequest(\_("Course id is invalid"))

if not has\_access(user, course, 'enroll'):

return HttpResponseBadRequest(\_("Enrollment is closed"))

# see if we have already filled up all allowed enrollments

is\_course\_full = CourseEnrollment.is\_course\_full(course)

if is\_course\_full:

return HttpResponseBadRequest(\_("Course is full"))

# If this course is available in multiple modes, redirect them to a page

# where they can choose which mode they want.

available\_modes = CourseMode.modes\_for\_course(course\_id)

if len(available\_modes) > 1:

return HttpResponse(

reverse("course\_modes\_choose", kwargs={'course\_id': course\_id})

)

current\_mode = available\_modes[0]

org, course\_num, run = course\_id.split("/")

dog\_stats\_api.increment(

"common.student.enrollment",

tags=[u"org:{0}".format(org),

u"course:{0}".format(course\_num),

u"run:{0}".format(run)]

)

CourseEnrollment.enroll(user, course.id, mode=current\_mode.slug)

return HttpResponse()

elif action == "add\_to\_cart":

# Pass the request handling to shoppingcart.views

# The view in shoppingcart.views performs error handling and logs different errors. But this elif clause

# is only used in the "auto-add after user reg/login" case, i.e. it's always wrapped in try\_change\_enrollment.

# This means there's no good way to display error messages to the user. So we log the errors and send

# the user to the shopping cart page always, where they can reasonably discern the status of their cart,

# whether things got added, etc

shoppingcart.views.add\_course\_to\_cart(request, course\_id)

return HttpResponse(

reverse("shoppingcart.views.show\_cart")

)

elif action == "unenroll":

if not CourseEnrollment.is\_enrolled(user, course\_id):

return HttpResponseBadRequest(\_("You are not enrolled in this course"))

CourseEnrollment.unenroll(user, course\_id)

org, course\_num, run = course\_id.split("/")

dog\_stats\_api.increment(

"common.student.unenrollment",

tags=["org:{0}".format(org),

"course:{0}".format(course\_num),

"run:{0}".format(run)]

)

return HttpResponse()

else:

return HttpResponseBadRequest(\_("Enrollment action is invalid"))

def \_parse\_course\_id\_from\_string(input\_str):

"""

Helper function to determine if input\_str (typically the queryparam 'next') contains a course\_id.

@param input\_str:

@return: the course\_id if found, None if not

"""

m\_obj = re.match(r'^/courses/(?P<course\_id>[^/]+/[^/]+/[^/]+)', input\_str)

if m\_obj:

return m\_obj.group('course\_id')

return None

def \_get\_course\_enrollment\_domain(course\_id):

"""

Helper function to get the enrollment domain set for a course with id course\_id

@param course\_id:

@return:

"""

try:

course = course\_from\_id(course\_id)

return course.enrollment\_domain

except ItemNotFoundError:

return None

@ensure\_csrf\_cookie

def accounts\_login(request):

"""

This view is mainly used as the redirect from the @login\_required decorator. I don't believe that

the login path linked from the homepage uses it.

"""

if settings.FEATURES.get('AUTH\_USE\_CAS'):

return redirect(reverse('cas-login'))

if settings.FEATURES['AUTH\_USE\_CERTIFICATES']:

# SSL login doesn't require a view, so redirect

# to branding and allow that to process the login.

return redirect(reverse('root'))

# see if the "next" parameter has been set, whether it has a course context, and if so, whether

# there is a course-specific place to redirect

redirect\_to = request.GET.get('next')

if redirect\_to:

course\_id = \_parse\_course\_id\_from\_string(redirect\_to)

if course\_id and \_get\_course\_enrollment\_domain(course\_id):

return external\_auth.views.course\_specific\_login(request, course\_id)

context = {

'platform\_name': settings.PLATFORM\_NAME,

}

return render\_to\_response('login.html', context)

# Need different levels of logging

@ensure\_csrf\_cookie

def login\_user(request, error=""):

"""AJAX request to log in the user."""

if 'email' not in request.POST or 'password' not in request.POST:

return JsonResponse({

"success": False,

"value": \_('There was an error receiving your login information. Please email us.'), # TODO: User error message

}) # TODO: this should be status code 400 # pylint: disable=fixme

email = request.POST['email']

password = request.POST['password']

try:

user = User.objects.get(email=email)

except User.DoesNotExist:

AUDIT\_LOG.warning(u"Login failed - Unknown user email: {0}".format(email))

user = None

# check if the user has a linked shibboleth account, if so, redirect the user to shib-login

# This behavior is pretty much like what gmail does for shibboleth. Try entering some @stanford.edu

# address into the Gmail login.

if settings.FEATURES.get('AUTH\_USE\_SHIB') and user:

try:

eamap = ExternalAuthMap.objects.get(user=user)

if eamap.external\_domain.startswith(external\_auth.views.SHIBBOLETH\_DOMAIN\_PREFIX):

return JsonResponse({

"success": False,

"redirect": reverse('shib-login'),

}) # TODO: this should be status code 301 # pylint: disable=fixme

except ExternalAuthMap.DoesNotExist:

# This is actually the common case, logging in user without external linked login

AUDIT\_LOG.info("User %s w/o external auth attempting login", user)

# see if account has been locked out due to excessive login failres

user\_found\_by\_email\_lookup = user

if user\_found\_by\_email\_lookup and LoginFailures.is\_feature\_enabled():

if LoginFailures.is\_user\_locked\_out(user\_found\_by\_email\_lookup):

return JsonResponse({

"success": False,

"value": \_('This account has been temporarily locked due to excessive login failures. Try again later.'),

}) # TODO: this should be status code 429 # pylint: disable=fixme

# if the user doesn't exist, we want to set the username to an invalid

# username so that authentication is guaranteed to fail and we can take

# advantage of the ratelimited backend

username = user.username if user else ""

try:

user = authenticate(username=username, password=password, request=request)

# this occurs when there are too many attempts from the same IP address

except RateLimitException:

return JsonResponse({

"success": False,

"value": \_('Too many failed login attempts. Try again later.'),

}) # TODO: this should be status code 429 # pylint: disable=fixme

if user is None:

# tick the failed login counters if the user exists in the database

if user\_found\_by\_email\_lookup and LoginFailures.is\_feature\_enabled():

LoginFailures.increment\_lockout\_counter(user\_found\_by\_email\_lookup)

# if we didn't find this username earlier, the account for this email

# doesn't exist, and doesn't have a corresponding password

if username != "":

AUDIT\_LOG.warning(u"Login failed - password for {0} is invalid".format(email))

return JsonResponse({

"success": False,

"value": \_('Email or password is incorrect.'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

# successful login, clear failed login attempts counters, if applicable

if LoginFailures.is\_feature\_enabled():

LoginFailures.clear\_lockout\_counter(user)

if user is not None and user.is\_active:

try:

# We do not log here, because we have a handler registered

# to perform logging on successful logins.

login(request, user)

if request.POST.get('remember') == 'true':

request.session.set\_expiry(604800)

log.debug("Setting user session to never expire")

else:

request.session.set\_expiry(0)

except Exception as e:

AUDIT\_LOG.critical("Login failed - Could not create session. Is memcached running?")

log.critical("Login failed - Could not create session. Is memcached running?")

log.exception(e)

raise

redirect\_url = try\_change\_enrollment(request)

dog\_stats\_api.increment("common.student.successful\_login")

response = JsonResponse({

"success": True,

"redirect\_url": redirect\_url,

})

# set the login cookie for the xuetangx marketing site

# we want this cookie to be accessed via javascript

# so httponly is set to None

if request.session.get\_expire\_at\_browser\_close():

max\_age = None

expires = None

else:

max\_age = request.session.get\_expiry\_age()

expires\_time = time.time() + max\_age

expires = cookie\_date(expires\_time)

response.set\_cookie(

settings.XUETANGXMKTG\_COOKIE\_NAME, 'true', max\_age=max\_age,

expires=expires, domain=settings.SESSION\_COOKIE\_DOMAIN,

path='/', secure=None, httponly=None,

)

return response

AUDIT\_LOG.warning(u"Login failed - Account not active for user {0}, resending activation".format(username))

reactivation\_email\_for\_user(user)

not\_activated\_msg = \_("This account has not been activated. We have sent another activation message. Please check your e-mail for the activation instructions.")

return JsonResponse({

"success": False,

"value": not\_activated\_msg,

}) # TODO: this should be status code 400 # pylint: disable=fixme

# Need different levels of logging

@csrf\_exempt

def auth\_user(request, error=""):

if 'security\_key' not in request.POST or request.POST['security\_key'] != 'tsinghuax':

return HttpResponse(json.dumps({'success': False,

'value': 'Security key error'}))

"""AJAX request to log in the user."""

if 'email' not in request.POST or 'password' not in request.POST:

return JsonResponse({

"success": False,

"value": \_('There was an error receiving your login information. Please email us.'), # TODO: User error message

}) # TODO: this should be status code 400 # pylint: disable=fixme

email = request.POST['email']

password = request.POST['password']

try:

user = User.objects.get(email=email)

except User.DoesNotExist:

AUDIT\_LOG.warning(u"Login failed - Unknown user email: {0}".format(email))

user = None

# check if the user has a linked shibboleth account, if so, redirect the user to shib-login

# This behavior is pretty much like what gmail does for shibboleth. Try entering some @stanford.edu

# address into the Gmail login.

if settings.FEATURES.get('AUTH\_USE\_SHIB') and user:

try:

eamap = ExternalAuthMap.objects.get(user=user)

if eamap.external\_domain.startswith(external\_auth.views.SHIBBOLETH\_DOMAIN\_PREFIX):

return JsonResponse({

"success": False,

"redirect": reverse('shib-login'),

}) # TODO: this should be status code 301 # pylint: disable=fixme

except ExternalAuthMap.DoesNotExist:

# This is actually the common case, logging in user without external linked login

AUDIT\_LOG.info("User %s w/o external auth attempting login", user)

# see if account has been locked out due to excessive login failres

user\_found\_by\_email\_lookup = user

if user\_found\_by\_email\_lookup and LoginFailures.is\_feature\_enabled():

if LoginFailures.is\_user\_locked\_out(user\_found\_by\_email\_lookup):

return JsonResponse({

"success": False,

"value": \_('This account has been temporarily locked due to excessive login failures. Try again later.'),

}) # TODO: this should be status code 429 # pylint: disable=fixme

# if the user doesn't exist, we want to set the username to an invalid

# username so that authentication is guaranteed to fail and we can take

# advantage of the ratelimited backend

username = user.username if user else ""

try:

user = authenticate(username=username, password=password, request=request)

# this occurs when there are too many attempts from the same IP address

except RateLimitException:

return JsonResponse({

"success": False,

"value": \_('Too many failed login attempts. Try again later.'),

}) # TODO: this should be status code 429 # pylint: disable=fixme

if user is None:

# tick the failed login counters if the user exists in the database

if user\_found\_by\_email\_lookup and LoginFailures.is\_feature\_enabled():

LoginFailures.increment\_lockout\_counter(user\_found\_by\_email\_lookup)

# if we didn't find this username earlier, the account for this email

# doesn't exist, and doesn't have a corresponding password

if username != "":

AUDIT\_LOG.warning(u"Login failed - password for {0} is invalid".format(email))

return JsonResponse({

"success": False,

"value": \_('Email or password is incorrect.'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

# successful login, clear failed login attempts counters, if applicable

if LoginFailures.is\_feature\_enabled():

LoginFailures.clear\_lockout\_counter(user)

if user is not None and user.is\_active:

try:

# We do not log here, because we have a handler registered

# to perform logging on successful logins.

login(request, user)

if request.POST.get('remember') == 'true':

request.session.set\_expiry(604800)

log.debug("Setting user session to never expire")

else:

request.session.set\_expiry(0)

except Exception as e:

AUDIT\_LOG.critical("Login failed - Could not create session. Is memcached running?")

log.critical("Login failed - Could not create session. Is memcached running?")

log.exception(e)

raise

redirect\_url = try\_change\_enrollment(request)

dog\_stats\_api.increment("common.student.successful\_login")

response = JsonResponse({

"success": True,

"redirect\_url": redirect\_url,

'email': user.email,

'username': user.username

})

# set the login cookie for the xuetangx marketing site

# we want this cookie to be accessed via javascript

# so httponly is set to None

if request.session.get\_expire\_at\_browser\_close():

max\_age = None

expires = None

else:

max\_age = request.session.get\_expiry\_age()

expires\_time = time.time() + max\_age

expires = cookie\_date(expires\_time)

response.set\_cookie(

settings.XUETANGXMKTG\_COOKIE\_NAME, 'true', max\_age=max\_age,

expires=expires, domain=settings.SESSION\_COOKIE\_DOMAIN,

path='/', secure=None, httponly=None,

)

return response

AUDIT\_LOG.warning(u"Login failed - Account not active for user {0}, resending activation".format(username))

reactivation\_email\_for\_user(user)

not\_activated\_msg = \_("This account has not been activated. We have sent another activation message. Please check your e-mail for the activation instructions.")

return JsonResponse({

"success": False,

"value": not\_activated\_msg,

}) # TODO: this should be status code 400 # pylint: disable=fixme

@ensure\_csrf\_cookie

def logout\_user(request):

"""

HTTP request to log out the user. Redirects to marketing page.

Deletes both the CSRF and sessionid cookies so the marketing

site can determine the logged in state of the user

"""

# We do not log here, because we have a handler registered

# to perform logging on successful logouts.

logout(request)

if settings.FEATURES.get('AUTH\_USE\_CAS'):

target = reverse('cas-logout')

else:

target = '/'

response = redirect(target)

response.delete\_cookie(

settings.XUETANGXMKTG\_COOKIE\_NAME,

path='/', domain=settings.SESSION\_COOKIE\_DOMAIN,

)

return response

@require\_GET

@login\_required

@ensure\_csrf\_cookie

def manage\_user\_standing(request):

"""

Renders the view used to manage user standing. Also displays a table

of user accounts that have been disabled and who disabled them.

"""

if not request.user.is\_staff:

raise Http404

all\_disabled\_accounts = UserStanding.objects.filter(

account\_status=UserStanding.ACCOUNT\_DISABLED

)

all\_disabled\_users = [standing.user for standing in all\_disabled\_accounts]

headers = ['username', 'account\_changed\_by']

rows = []

for user in all\_disabled\_users:

row = [user.username, user.standing.all()[0].changed\_by]

rows.append(row)

context = {'headers': headers, 'rows': rows}

return render\_to\_response("manage\_user\_standing.html", context)

@require\_POST

@login\_required

@ensure\_csrf\_cookie

def disable\_account\_ajax(request):

"""

Ajax call to change user standing. Endpoint of the form

in manage\_user\_standing.html

"""

if not request.user.is\_staff:

raise Http404

username = request.POST.get('username')

context = {}

if username is None or username.strip() == '':

context['message'] = \_('Please enter a username')

return JsonResponse(context, status=400)

account\_action = request.POST.get('account\_action')

if account\_action is None:

context['message'] = \_('Please choose an option')

return JsonResponse(context, status=400)

username = username.strip()

try:

user = User.objects.get(username=username)

except User.DoesNotExist:

context['message'] = \_("User with username {} does not exist").format(username)

return JsonResponse(context, status=400)

else:

user\_account, \_success = UserStanding.objects.get\_or\_create(

user=user, defaults={'changed\_by': request.user},

)

if account\_action == 'disable':

user\_account.account\_status = UserStanding.ACCOUNT\_DISABLED

context['message'] = \_("Successfully disabled {}'s account").format(username)

log.info("{} disabled {}'s account".format(request.user, username))

elif account\_action == 'reenable':

user\_account.account\_status = UserStanding.ACCOUNT\_ENABLED

context['message'] = \_("Successfully reenabled {}'s account").format(username)

log.info("{} reenabled {}'s account".format(request.user, username))

else:

context['message'] = \_("Unexpected account status")

return JsonResponse(context, status=400)

user\_account.changed\_by = request.user

user\_account.standing\_last\_changed\_at = datetime.datetime.now(UTC)

user\_account.save()

return JsonResponse(context)

@login\_required

@ensure\_csrf\_cookie

def change\_setting(request):

"""JSON call to change a profile setting: Right now, location"""

# TODO (vshnayder): location is no longer used

# request.user.profile\_cache

up = UserProfile.objects.get(user=request.user)

if 'location' in request.POST:

up.location = request.POST['location']

up.save()

return JsonResponse({

"success": True,

"location": up.location,

})

def check\_email\_available(request):

get\_vars = request.GET

js = {'success': False}

if len(User.objects.filter(email=get\_vars['email'])) > 0:

js['value'] = \_("An account with the Email '{email}' already exists.").format(email=get\_vars['email'])

js['field'] = 'email'

return JsonResponse(js, status=200)

js['success'] = True

return JsonResponse(js, status=200)

def check\_username\_available(request):

get\_vars = request.GET

js = {'success': False}

if len(User.objects.filter(username=get\_vars['username'])) > 0:

js['value'] = \_("An account with the Public Username '{username}' already exists.").format(username=get\_vars['username'])

js['field'] = 'username'

return JsonResponse(js, status=200)

js['success'] = True

return JsonResponse(js, status=200)

def \_do\_create\_account(post\_vars, is\_active=False):

"""

Given cleaned post variables, create the User and UserProfile objects, as well as the

registration for this user.

Returns a tuple (User, UserProfile, Registration).

Note: this function is also used for creating test users.

"""

user = User(username=post\_vars['username'],

email=post\_vars['email'],

is\_active=is\_active)

user.set\_password(post\_vars['password'])

registration = Registration()

# TODO: Rearrange so that if part of the process fails, the whole process fails.

# Right now, we can have e.g. no registration e-mail sent out and a zombie

# account

try:

user.save()

except IntegrityError:

js = {'success': False}

# Figure out the cause of the integrity error

if len(User.objects.filter(username=post\_vars['username'])) > 0:

js['value'] = \_("An account with the Public Username '{username}' already exists.").format(username=post\_vars['username'])

js['field'] = 'username'

return JsonResponse(js, status=400)

if len(User.objects.filter(email=post\_vars['email'])) > 0:

js['value'] = \_("An account with the Email '{email}' already exists.").format(email=post\_vars['email'])

js['field'] = 'email'

return JsonResponse(js, status=400)

raise

registration.register(user)

profile = UserProfile(user=user)

profile.name = post\_vars['name']

profile.level\_of\_education = post\_vars.get('level\_of\_education')

profile.gender = post\_vars.get('gender')

profile.mailing\_address = post\_vars.get('mailing\_address')

profile.city = post\_vars.get('city')

profile.country = post\_vars.get('country')

profile.goals = post\_vars.get('goals')

try:

profile.year\_of\_birth = int(post\_vars['year\_of\_birth'])

except (ValueError, KeyError):

# If they give us garbage, just ignore it instead

# of asking them to put an integer.

profile.year\_of\_birth = None

try:

profile.save()

except Exception:

log.exception("UserProfile creation failed for user {id}.".format(id=user.id))

return (user, profile, registration)

@ensure\_csrf\_cookie

def create\_account(request, post\_override=None):

"""

JSON call to create new xuetangx account.

Used by form in signup\_modal.html, which is included into navigation.html

"""

js = {'success': False}

post\_vars = post\_override if post\_override else request.POST

extra\_fields = getattr(settings, 'REGISTRATION\_EXTRA\_FIELDS', {})

# if doing signup for an external authorization, then get email, password, name from the eamap

# don't use the ones from the form, since the user could have hacked those

# unless originally we didn't get a valid email or name from the external

# auth

DoExternalAuth = 'ExternalAuthMap' in request.session

if DoExternalAuth:

eamap = request.session['ExternalAuthMap']

try:

validate\_email(eamap.external\_email)

email = eamap.external\_email

except ValidationError:

email = post\_vars.get('email', '')

if eamap.external\_name.strip() == '':

name = post\_vars.get('name', '')

else:

name = eamap.external\_name

password = eamap.internal\_password

post\_vars = dict(post\_vars.items())

post\_vars.update(dict(email=email, name=name, password=password))

log.debug(u'In create\_account with external\_auth: user = %s, email=%s', name, email)

# Confirm we have a properly formed request

for a in ['username', 'email', 'password', 'name']:

if a not in post\_vars:

js['value'] = \_("Error (401 {field}). E-mail us.").format(field=a)

js['field'] = a

return JsonResponse(js, status=400)

if extra\_fields.get('honor\_code', 'required') == 'required' and \

post\_vars.get('honor\_code', 'false') != u'true':

js['value'] = \_("To enroll, you must follow the honor code.").format(field=a)

js['field'] = 'honor\_code'

return JsonResponse(js, status=400)

# Can't have terms of service for certain SHIB users, like at Stanford

tos\_required = (

not settings.FEATURES.get("AUTH\_USE\_SHIB") or

not settings.FEATURES.get("SHIB\_DISABLE\_TOS") or

not DoExternalAuth or

not eamap.external\_domain.startswith(

external\_auth.views.SHIBBOLETH\_DOMAIN\_PREFIX

)

)

if tos\_required:

if post\_vars.get('terms\_of\_service', 'false') != u'true':

js['value'] = \_(

"You must accept the terms of service.").format(field=a)

js['field'] = 'terms\_of\_service'

return JsonResponse(js, status=400)

# Confirm appropriate fields are there.

# TODO: Check e-mail format is correct.

# TODO: Confirm e-mail is not from a generic domain (mailinator, etc.)? Not sure if

# this is a good idea

# TODO: Check password is sane

required\_post\_vars = ['username', 'email', 'name', 'password']

required\_post\_vars += [fieldname for fieldname, val in extra\_fields.items()

if val == 'required']

if tos\_required:

required\_post\_vars.append('terms\_of\_service')

for field\_name in required\_post\_vars:

if field\_name in ('gender', 'level\_of\_education'):

min\_length = 1

else:

min\_length = 2

if len(post\_vars[field\_name]) < min\_length:

error\_str = {

'username': \_('Username must be minimum of two characters long'),

'email': \_('A properly formatted e-mail is required'),

'name': \_('Your legal name must be a minimum of two characters long'),

'password': \_('A valid password is required'),

'terms\_of\_service': \_('Accepting Terms of Service is required'),

'honor\_code': \_('Agreeing to the Honor Code is required'),

'level\_of\_education': \_('A level of education is required'),

'gender': \_('Your gender is required'),

'year\_of\_birth': \_('Your year of birth is required'),

'mailing\_address': \_('Your mailing address is required'),

'goals': \_('A description of your goals is required'),

'city': \_('A city is required'),

'country': \_('A country is required')

}

js['value'] = error\_str[field\_name]

js['field'] = field\_name

return JsonResponse(js, status=400)

try:

validate\_email(post\_vars['email'])

except ValidationError:

js['value'] = \_("Valid e-mail is required.").format(field=a)

js['field'] = 'email'

return JsonResponse(js, status=400)

if post\_vars['repassword'] != post\_vars['password']:

js['value'] = \_('Enter the password twice to the same')

js['field'] = 'repassword'

return JsonResponse(js, status=400)

try:

validate\_slug(post\_vars['username'])

except ValidationError:

js['value'] = \_(

'Username should only consist of A-Z and 0-9 and "\_" and "-", with no spaces.').format(field=a)

js['field'] = 'username'

return JsonResponse(js, status=400)

# enforce password complexity as an optional feature

if settings.FEATURES.get('ENFORCE\_PASSWORD\_POLICY', False):

try:

password = post\_vars['password']

validate\_password\_length(password)

validate\_password\_complexity(password)

validate\_password\_dictionary(password)

except ValidationError, err:

js['value'] = \_('Password: ') + '; '.join(err.messages)

js['field'] = 'password'

return JsonResponse(js, status=400)

# Ok, looks like everything is legit. Create the account.

ret = \_do\_create\_account(post\_vars)

if isinstance(ret, HttpResponse): # if there was an error then return that

return ret

(user, profile, registration) = ret

context = {

'name': post\_vars['name'],

'key': registration.activation\_key,

}

# composes activation email

subject = render\_to\_string('emails/activation\_email\_subject.txt', context)

# Email subject \*must not\* contain newlines

subject = ''.join(subject.splitlines())

message = render\_to\_string('emails/activation\_email.txt', context)

# don't send email if we are doing load testing or random user generation for some reason

if not (settings.FEATURES.get('AUTOMATIC\_AUTH\_FOR\_TESTING')):

from\_address = MicrositeConfiguration.get\_microsite\_configuration\_value(

'email\_from\_address',

settings.DEFAULT\_FROM\_EMAIL

)

try:

if settings.FEATURES.get('REROUTE\_ACTIVATION\_EMAIL'):

dest\_addr = settings.FEATURES['REROUTE\_ACTIVATION\_EMAIL']

message = ("Activation for %s (%s): %s\n" % (user, user.email, profile.name) +

'-' \* 80 + '\n\n' + message)

send\_mail(subject, message, from\_address, [dest\_addr], fail\_silently=False)

else:

user.email\_user(subject, message, from\_address)

except Exception: # pylint: disable=broad-except

log.warning('Unable to send activation email to user', exc\_info=True)

js['value'] = \_('Could not send activation e-mail.')

# What is the correct status code to use here? I think it's 500, because

# the problem is on the server's end -- but also, the account was created.

# Seems like the core part of the request was successful.

return JsonResponse(js, status=500)

# Immediately after a user creates an account, we log them in. They are only

# logged in until they close the browser. They can't log in again until they click

# the activation link from the email.

login\_user = authenticate(

username=post\_vars['username'], password=post\_vars['password'])

login(request, login\_user)

request.session.set\_expiry(0)

# TODO: there is no error checking here to see that the user actually logged in successfully,

# and is not yet an active user.

if login\_user is not None:

AUDIT\_LOG.info(u"Login success on new account creation - {0}".format(login\_user.username))

if DoExternalAuth:

eamap.user = login\_user

eamap.dtsignup = datetime.datetime.now(UTC)

eamap.save()

AUDIT\_LOG.info("User registered with external\_auth %s", post\_vars['username'])

AUDIT\_LOG.info('Updated ExternalAuthMap for %s to be %s', post\_vars['username'], eamap)

if settings.FEATURES.get('BYPASS\_ACTIVATION\_EMAIL\_FOR\_EXTAUTH'):

log.info('bypassing activation email')

login\_user.is\_active = True

login\_user.save()

AUDIT\_LOG.info(u"Login activated on extauth account - {0} ({1})".format(login\_user.username, login\_user.email))

dog\_stats\_api.increment("common.student.account\_created")

response = JsonResponse({

'success': True,

'redirect\_url': try\_change\_enrollment(request),

})

# set the login cookie for the xuetangx marketing site

# we want this cookie to be accessed via javascript

# so httponly is set to None

if request.session.get\_expire\_at\_browser\_close():

max\_age = None

expires = None

else:

max\_age = request.session.get\_expiry\_age()

expires\_time = time.time() + max\_age

expires = cookie\_date(expires\_time)

response.set\_cookie(settings.XUETANGXMKTG\_COOKIE\_NAME,

'true', max\_age=max\_age,

expires=expires, domain=settings.SESSION\_COOKIE\_DOMAIN,

path='/',

secure=None,

httponly=None)

return response

def auto\_auth(request):

"""

Create or configure a user account, then log in as that user.

Enabled only when

settings.FEATURES['AUTOMATIC\_AUTH\_FOR\_TESTING'] is true.

Accepts the following querystring parameters:

\* `username`, `email`, and `password` for the user account

\* `full\_name` for the user profile (the user's full name; defaults to the username)

\* `staff`: Set to "true" to make the user global staff.

\* `course\_id`: Enroll the student in the course with `course\_id`

If username, email, or password are not provided, use

randomly generated credentials.

"""

# Generate a unique name to use if none provided

unique\_name = uuid.uuid4().hex[0:30]

# Use the params from the request, otherwise use these defaults

username = request.GET.get('username', unique\_name)

password = request.GET.get('password', unique\_name)

email = request.GET.get('email', unique\_name + "@example.com")

full\_name = request.GET.get('full\_name', username)

is\_staff = request.GET.get('staff', None)

course\_id = request.GET.get('course\_id', None)

# Get or create the user object

post\_data = {

'username': username,

'email': email,

'password': password,

'name': full\_name,

'honor\_code': u'true',

'terms\_of\_service': u'true',

}

# Attempt to create the account.

# If successful, this will return a tuple containing

# the new user object; otherwise it will return an error

# message.

result = \_do\_create\_account(post\_data)

if isinstance(result, tuple):

user = result[0]

# If we did not create a new account, the user might already

# exist. Attempt to retrieve it.

else:

user = User.objects.get(username=username)

user.email = email

user.set\_password(password)

user.save()

# Set the user's global staff bit

if is\_staff is not None:

user.is\_staff = (is\_staff == "true")

user.save()

# Activate the user

reg = Registration.objects.get(user=user)

reg.activate()

reg.save()

# Enroll the user in a course

if course\_id is not None:

CourseEnrollment.enroll(user, course\_id)

# Log in as the user

user = authenticate(username=username, password=password)

login(request, user)

# Provide the user with a valid CSRF token

# then return a 200 response

success\_msg = u"Logged in user {0} ({1}) with password {2}".format(

username, email, password

)

response = HttpResponse(success\_msg)

response.set\_cookie('csrftoken', csrf(request)['csrf\_token'])

return response

@ensure\_csrf\_cookie

def activate\_account(request, key):

"""When link in activation e-mail is clicked"""

r = Registration.objects.filter(activation\_key=key)

if len(r) == 1:

user\_logged\_in = request.user.is\_authenticated()

already\_active = True

if not r[0].user.is\_active:

r[0].activate()

already\_active = False

# Enroll student in any pending courses he/she may have if auto\_enroll flag is set

student = User.objects.filter(id=r[0].user\_id)

if student:

ceas = CourseEnrollmentAllowed.objects.filter(

email=student[0].email)

for cea in ceas:

if cea.auto\_enroll:

CourseEnrollment.enroll(student[0], cea.course\_id)

resp = render\_to\_response(

"registration/activation\_complete.html",

{

'user\_logged\_in': user\_logged\_in,

'already\_active': already\_active

}

)

return resp

if len(r) == 0:

return render\_to\_response(

"registration/activation\_invalid.html",

{'csrf': csrf(request)['csrf\_token']}

)

return HttpResponse(\_("Unknown error. Please e-mail us to let us know how it happened."))

@ensure\_csrf\_cookie

def password\_reset(request):

""" Attempts to send a password reset e-mail. """

if request.method != "POST":

raise Http404

form = PasswordResetFormNoActive(request.POST)

if form.is\_valid():

form.save(use\_https=request.is\_secure(),

from\_email=settings.DEFAULT\_FROM\_EMAIL,

request=request,

domain\_override=request.get\_host())

return JsonResponse({

'success': True,

'value': render\_to\_string('registration/password\_reset\_done.html', {}),

})

def password\_reset\_confirm\_wrapper(

request,

uidb36=None,

token=None,

):

""" A wrapper around django.contrib.auth.views.password\_reset\_confirm.

Needed because we want to set the user as active at this step.

"""

# cribbed from django.contrib.auth.views.password\_reset\_confirm

try:

uid\_int = base36\_to\_int(uidb36)

user = User.objects.get(id=uid\_int)

user.is\_active = True

user.save()

except (ValueError, User.DoesNotExist):

pass

# we also want to pass settings.PLATFORM\_NAME in as extra\_context

extra\_context = {"platform\_name": settings.PLATFORM\_NAME}

return password\_reset\_confirm(

request, uidb36=uidb36, token=token, extra\_context=extra\_context

)

def reactivation\_email\_for\_user(user):

try:

reg = Registration.objects.get(user=user)

except Registration.DoesNotExist:

return JsonResponse({

"success": False,

"error": \_('No inactive user with this e-mail exists'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

context = {

'name': user.profile.name,

'key': reg.activation\_key,

}

subject = render\_to\_string('emails/activation\_email\_subject.txt', context)

subject = ''.join(subject.splitlines())

message = render\_to\_string('emails/activation\_email.txt', context)

try:

user.email\_user(subject, message, settings.DEFAULT\_FROM\_EMAIL)

except Exception: # pylint: disable=broad-except

log.warning('Unable to send reactivation email', exc\_info=True)

return JsonResponse({

"success": False,

"error": \_('Unable to send reactivation email')

}) # TODO: this should be status code 500 # pylint: disable=fixme

return JsonResponse({"success": True})

@ensure\_csrf\_cookie

def change\_email\_request(request):

""" AJAX call from the profile page. User wants a new e-mail.

"""

## Make sure it checks for existing e-mail conflicts

if not request.user.is\_authenticated:

raise Http404

user = request.user

if not user.check\_password(request.POST['password']):

return JsonResponse({

"success": False,

"error": \_('Invalid password'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

new\_email = request.POST['new\_email']

try:

validate\_email(new\_email)

except ValidationError:

return JsonResponse({

"success": False,

"error": \_('Valid e-mail address required.'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

if User.objects.filter(email=new\_email).count() != 0:

## CRITICAL TODO: Handle case sensitivity for e-mails

return JsonResponse({

"success": False,

"error": \_('An account with this e-mail already exists.'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

pec\_list = PendingEmailChange.objects.filter(user=request.user)

if len(pec\_list) == 0:

pec = PendingEmailChange()

pec.user = user

else:

pec = pec\_list[0]

pec.new\_email = request.POST['new\_email']

pec.activation\_key = uuid.uuid4().hex

pec.save()

if pec.new\_email == user.email:

pec.delete()

return JsonResponse({

"success": False,

"error": \_('Old email is the same as the new email.'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

context = {

'key': pec.activation\_key,

'old\_email': user.email,

'new\_email': pec.new\_email

}

subject = render\_to\_string('emails/email\_change\_subject.txt', context)

subject = ''.join(subject.splitlines())

message = render\_to\_string('emails/email\_change.txt', context)

from\_address = MicrositeConfiguration.get\_microsite\_configuration\_value(

'email\_from\_address',

settings.DEFAULT\_FROM\_EMAIL

)

send\_mail(subject, message, from\_address, [pec.new\_email])

return JsonResponse({"success": True})

@ensure\_csrf\_cookie

@transaction.commit\_manually

def confirm\_email\_change(request, key):

""" User requested a new e-mail. This is called when the activation

link is clicked. We confirm with the old e-mail, and update

"""

try:

try:

pec = PendingEmailChange.objects.get(activation\_key=key)

except PendingEmailChange.DoesNotExist:

response = render\_to\_response("invalid\_email\_key.html", {})

transaction.rollback()

return response

user = pec.user

address\_context = {

'old\_email': user.email,

'new\_email': pec.new\_email

}

if len(User.objects.filter(email=pec.new\_email)) != 0:

response = render\_to\_response("email\_exists.html", {})

transaction.rollback()

return response

subject = render\_to\_string(

'emails/email\_change\_subject.txt', address\_context)

subject = ''.join(subject.splitlines())

message = render\_to\_string(

'emails/confirm\_email\_change.txt', address\_context)

up = UserProfile.objects.get(user=user)

meta = up.get\_meta()

if 'old\_emails' not in meta:

meta['old\_emails'] = []

meta['old\_emails'].append([user.email, datetime.datetime.now(UTC).isoformat()])

up.set\_meta(meta)

up.save()

# Send it to the old email...

try:

user.email\_user(subject, message, settings.DEFAULT\_FROM\_EMAIL)

except Exception:

log.warning('Unable to send confirmation email to old address', exc\_info=True)

response = render\_to\_response("email\_change\_failed.html", {'email': user.email})

transaction.rollback()

return response

user.email = pec.new\_email

user.save()

pec.delete()

# And send it to the new email...

try:

user.email\_user(subject, message, settings.DEFAULT\_FROM\_EMAIL)

except Exception:

log.warning('Unable to send confirmation email to new address', exc\_info=True)

response = render\_to\_response("email\_change\_failed.html", {'email': pec.new\_email})

transaction.rollback()

return response

response = render\_to\_response("email\_change\_successful.html", address\_context)

transaction.commit()

return response

except Exception:

# If we get an unexpected exception, be sure to rollback the

# transaction

transaction.rollback()

raise

@ensure\_csrf\_cookie

def change\_name\_request(request):

""" Log a request for a new name. """

if not request.user.is\_authenticated:

raise Http404

try:

pnc = PendingNameChange.objects.get(user=request.user)

except PendingNameChange.DoesNotExist:

pnc = PendingNameChange()

pnc.user = request.user

pnc.new\_name = request.POST['new\_name']

pnc.rationale = request.POST['rationale']

if len(pnc.new\_name) < 2:

return JsonResponse({

"success": False,

"error": \_('Name required'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

pnc.save()

# The following automatically accepts name change requests. Remove this to

# go back to the old system where it gets queued up for admin approval.

accept\_name\_change\_by\_id(pnc.id)

return JsonResponse({"success": True})

@ensure\_csrf\_cookie

def pending\_name\_changes(request):

""" Web page which allows staff to approve or reject name changes. """

if not request.user.is\_staff:

raise Http404

students = []

for change in PendingNameChange.objects.all():

profile = UserProfile.objects.get(user=change.user)

students.append({

"new\_name": change.new\_name,

"rationale": change.rationale,

"old\_name": profile.name,

"email": change.user.email,

"uid": change.user.id,

"cid": change.id,

})

return render\_to\_response("name\_changes.html", {"students": students})

@ensure\_csrf\_cookie

def reject\_name\_change(request):

""" JSON: Name change process. Course staff clicks 'reject' on a given name change """

if not request.user.is\_staff:

raise Http404

try:

pnc = PendingNameChange.objects.get(id=int(request.POST['id']))

except PendingNameChange.DoesNotExist:

return JsonResponse({

"success": False,

"error": \_('Invalid ID'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

pnc.delete()

return JsonResponse({"success": True})

def accept\_name\_change\_by\_id(id):

try:

pnc = PendingNameChange.objects.get(id=id)

except PendingNameChange.DoesNotExist:

return JsonResponse({

"success": False,

"error": \_('Invalid ID'),

}) # TODO: this should be status code 400 # pylint: disable=fixme

u = pnc.user

up = UserProfile.objects.get(user=u)

# Save old name

meta = up.get\_meta()

if 'old\_names' not in meta:

meta['old\_names'] = []

meta['old\_names'].append([up.name, pnc.rationale, datetime.datetime.now(UTC).isoformat()])

up.set\_meta(meta)

up.name = pnc.new\_name

up.save()

pnc.delete()

return JsonResponse({"success": True})

@ensure\_csrf\_cookie

def accept\_name\_change(request):

""" JSON: Name change process. Course staff clicks 'accept' on a given name change

We used this during the prototype but now we simply record name changes instead

of manually approving them. Still keeping this around in case we want to go

back to this approval method.

"""

if not request.user.is\_staff:

raise Http404

return accept\_name\_change\_by\_id(int(request.POST['id']))

@require\_POST

@login\_required

@ensure\_csrf\_cookie

def change\_email\_settings(request):

"""Modify logged-in user's setting for receiving emails from a course."""

user = request.user

course\_id = request.POST.get("course\_id")

receive\_emails = request.POST.get("receive\_emails")

if receive\_emails:

optout\_object = Optout.objects.filter(user=user, course\_id=course\_id)

if optout\_object:

optout\_object.delete()

log.info(u"User {0} ({1}) opted in to receive emails from course {2}".format(user.username, user.email, course\_id))

track.views.server\_track(request, "change-email-settings", {"receive\_emails": "yes", "course": course\_id}, page='dashboard')

else:

Optout.objects.get\_or\_create(user=user, course\_id=course\_id)

log.info(u"User {0} ({1}) opted out of receiving emails from course {2}".format(user.username, user.email, course\_id))

track.views.server\_track(request, "change-email-settings", {"receive\_emails": "no", "course": course\_id}, page='dashboard')

return JsonResponse({"success": True})

@login\_required

def token(request):

'''

Return a token for the backend of annotations.

It uses the course id to retrieve a variable that contains the secret

token found in inheritance.py. It also contains information of when

the token was issued. This will be stored with the user along with

the id for identification purposes in the backend.

'''

course\_id = request.GET.get("course\_id")

course = course\_from\_id(course\_id)

dtnow = datetime.datetime.now()

dtutcnow = datetime.datetime.utcnow()

delta = dtnow - dtutcnow

newhour, newmin = divmod((delta.days \* 24 \* 60 \* 60 + delta.seconds + 30) // 60, 60)

newtime = "%s%+02d:%02d" % (dtnow.isoformat(), newhour, newmin)

secret = course.annotation\_token\_secret

custom\_data = {"issuedAt": newtime, "consumerKey": secret, "userId": request.user.email, "ttl": 86400}

newtoken = create\_token(secret, custom\_data)

response = HttpResponse(newtoken, mimetype="text/plain")

return response