Streamwebs Documentation

Release 1.0

Kenneth Lett

Jan 03, 2018

Contents

CHAPTER 1

General Documentation

1.1 About

StreamWebs is a dynamic networking platform that links students with locally based hands-on watershed stewardship projects and provides a multimedia showcase for their project and data reports. StreamWebs offers teachers and community partners resources to support setting students on the path to lifelong watershed stewardship. By providing students and teachers open-source, web-based tools for watershed data management, analysis, and networking, StreamWebs supports classrooms in their pursuit of STEM (Science, Technology, Engineering, Mathematics) educational opportunities, while helping students and teachers demonstrate their role as vital contributors to watershed sustainability. StreamWebs originated in 2008 as a project of The Freshwater Trust. In 2011, StreamWebs transitioned to Oregon State University Extension Service.

Oregon State University's Open Source Lab is currently remaking the site.

For more information, please visit the original StreamWebs site here, or send an email to StreamWebs@oregonstate.edu.

CHAPTER 2

User Documentation

2.1 Using StreamWebs

(Note: This user guide will updated accordingly as the StreamWebs website is built.)

StreamWebs allows you to log and browse data for various watershed sites. You don't need an account to view data, but in order to add a new site or record your own data on StreamWebs, you do need to register. Here's how to create an account and get started with StreamWebs.

2.1.1 Creating an Account

Click the "Create Account" tab located near the head of the StreamWebs home page. You will be taken to a form which will ask you to provide the following:

- Your date of birth
- Your desired username
- Your desired password
- Your email address
- Your first and last name
- The name of your school

You can indicate your school by typing all or part or its name into the appropriate field. A drop down menu will appear as you type and provide you with a list of schools that match what you have typed. You can then select your school from there. If you don't see your school listed in the drop down menu, first make sure that you are typing the official name of your school into the field rather than an abbreviation or nickname– for example, you should type "Oregon State University" instead of "OSU". If your school still fails to appear, you can select the "Other" option at the bottom of the list and enter the name manually.

When you're done with the form, hit the "Create New Account" button at the bottom of the page to complete your registration.

2.1.2 Logging In

After your StreamWebs account is set up, you can log in by clicking the "Log In" link at the top of the homepage and providing your password and username (or, alternatively, your email address). If you've forgotten your password, you can get a new one by clicking the "Request New Password" link. An email will be sent to you with instructions on how to reset your password.

2.1.3 Project Sites

Now that you're logged in, you can not only view data for sites, but contribute to them as well.

1. Viewing a Site

To browse sites, navigate to the "Project Sites" tab at the top of the homepage (after logging in, you will be taken here automatically). You can find a specific site by:

1. Typing all or part of the site's name into the search bar
2. Clicking a location marker on the map
3. Selecting the site from the pull down menu

Upon selection, you'll be taken to that site's page, where you can view its details (name, type, description, images, etc.). Under "Data" is a series of checkboxes generated from the site's available data sheets (for more on data sheets, see the "Resources" section of this document). When you select one of these fields, you can:

1. View the site's compiled data in a table, or
2. View the site's compiled data as a graph, or
3. Export the data as CSV file, or
4. Compare data between multiple sites.

Comparing Data

The "Compare" button on each site's page allows you to compare specific categories of data across multiple sites, or with outside resources online.

1. In the "Compare" section of your current site's page, indicate which type of data you'd like to examine. Then click "Compare."
2. This takes you to a new page with a list of sites that have the category of data you want. Select the sites you'd like to compare your data with.
3. Hit "Compare" again to generate graphs from each site on the same screen.

Site Stewards

Clicking "Site Stewards" on a site's page will display a list of the schools that have contributed to that site's data record. To filter site data by school, select your desired school and type of data. Then you click "Graph," "Table", or "Export" to graph that school's data, table that school's data, or export that school's data into a downloadable file, respectively.

2. Creating a New Site

From the central Sites page, you can click “Create” (note that you must be logged in). This takes you to a form where you can enter your site’s location, type, and description. Submit the form to add your new site to StreamWebs.

3. Recording Site Data

To start entering data, click “Add Data” from the central Sites page. You will be asked to select the school you’re from, the class you’re in, the site you visited, the date you visited the site, and the data sheet template that suits your needs.

2.1.4 Resources

Datasheets

CHAPTER 3

Development Documentation:

3.1 Overview

Streamwebs is an science education platform for educators and students in Oregon's public schools. The application allows students to enter data about streams and wetlands gathered on field trips, then graph or export the data for further analysis.

This application is a re-write of the original Drupal-based Streamwebs application. Until the new application goes live, the original can be found at Streamwebs.org.

3.1.1 Core Components

Streamwebs is written in Python usign the Django web framework. It follows Python's PEP8 coding style and standard Django MVC structure. Streamwebs also makes use of geographic information and the Google Maps API. The underlying data store is Postgres with geographic data extensions (PostGIS).

Streamwebs also uses the D3 javascript library for generating graphs, and the Materialize CSS framework for the front-end UI elements.

Django Structure

Django's MVC model is slightly odd, what would normally be called a Controller is a View, and what would be called a View is a Template. Models are what you would expect, a description of the data model, for use with Django's ORM. URLs are defined in a separate file.

3.1.2 Models

All models are defined in `streamwebs_frontend/streamwebs/models.py`

3.1.3 Views

Views are typically divided logically into several files according to the model they refer to. Streamwebs doesn't do this, all general views are defined in `streamwebs_frontend/streamwebs/views/general.py`, and a separate set of views specifically dealing with CSV export are in `streamwebs_frontend/streamwebs/views/export.py`

Views recieve the current http request object object and returns an HTTP response object. See Django Request and response objects for their structure.

Views may also be 'decorated' to restrict access to logged in users. See Django View Decorators for information on what can be done with decorators.

For example, you will see this construction frequently in the views file:

```
@login_required
def create_site(request):
    created = False
    ...
```

The *@login_required* is a decorator provided by the Django Authentication Framework and prevents non-authenticated users from accessing this view.

3.1.4 Templates and TemplateTags

Django by default uses a template language similar to Jinja2, other backends can be defined, but Streamwebs sticks to the default. Templates have the *.html* extension and are found in `streamwebs_frontend/streamwebs/templates/`

Templates have an inheritance structure, our base template sets up the HTML page, and other templates inherit this structure. See Django Templates for more information.

Streamwebs also makes use of TemplateTags, which can be used to transform text in templates. These are defined in `streamwebs_frontend/streamwebs/templatetags/filters.py`. See Custom Django Template Tags for more information.

In templates, filters are included using `{% load filters %}` at the topof the file. A filter is applied with the `|` character appended to a rendered template variable:

```
{{ 'zone'|get_zone_labels }}
```

3.1.5 URLs

Streamwebs defines its URLs in `streamwebs_frontend/streamwebs/urls.py`.

This file defines all URLs and connects them to their view method, which should then render the appropriate template.

URLs are namespaced, the `app_name = 'streamwebs'` line ensures that URLs are relative to the `streamwebs` app. In practicat terms, the important thing to know is that links in templates are specified with the streamwebs namespace:

```
{% url 'streamwebs:register' %}
```

This contruction will render the URL named `register` in the `streamwebs` application.

Sites

The Primary object in Streamwebs is the Site, which consists of a name, description, location (coordinates), and optionally an image. Site locations are point objects and are displayed in a map view as markers.

All data is associated with a site, and the site view is the starting point for entering, viewing, graphing or downloading collected data.

See *Sites* for more details.

Datasheets

Datasheets are associated with a site, and also with a school. They store collected data about a stream - air temperature, water temperature, species detected, etc.

There are six datasheets currently defined:

- Canopy Cover: records percentage of the sky obscured by forest canopy
- Macroinvertibrates: records the numbers of specific species of invertebrates
- Photo Point: records images taken at a specific location
- Riparian Transect: records plant types in a cross-section of a streambed
- Soil Survey: records the types of soil found
- Water Quality: records various water measurments such as temperature and oxygen level

Each datasheet has its own model, form, view and database table, and may have associated models, for example mulitple samples for a specific measurement.

Datasheet views are designed to resemble as closely as possible the paper versions that are taken into the field to record data. Calculated values are automatically calculated on data entry. PDF files for each sheet are provided for printing.

See *Datasheets* for more details.

Graphing

See *Graphing* for more details.

Users

Students and teachers share a general ‘user’ role. Users are authorized to enter data and create sites, the only reserved permissions are resource file uploads and viewing site statistics.

Users are associated with a School. Birth date is a required field for account creation, and students must be 13 years of age or older to sign up.

Users can self-register, by default they will be placed in the general user role, a admin can promote any user to an admin role. The initial admin account is the Django ‘superadmin’, created on deployment, and this superadmin should designate one or more user accounts as admins.

See *Users* for more details.

Resources

Several types of files are available for download from the site. A generic 'resource' model is used to store datasheets, educator kits, publications and tutorial videos, and these resources are displayed on type-specific pages. An index of all resources is located at /resources.

See *Resources* for more details.

Statistics

Basic site statistics are available to admin users. Number of users, site and datasheets are the primary statistics.

See *Statistics* for more details.

Schools

A model containing a list of known public schools in Oregon, used for tracking which school contributed data to a site. Also associated with users.

Note: Data is associeted explicitly with a school, we do not rely on the account of the user who entered data for determining which school created the data.

See *Schools* for more details.

Test Suite

All views, forms, permissions and models are tested with unit tests. These tests use the Django test framework.

In addition to tests, files should be analyzed by the Flake8 python linter, which enforces Python standard PEP8.

Development of new features should begin with writing a test for that feature.

See *Tests* for more details.

Internationalization

Streamwebs uses the Django translation framework to translate strings into supported languages. Supported languages can be selected using a pull-down selector in the application.

Supported languages are set using the `LANGUAGES` setting. The default application application language is set in `LANGUAGE_CODE` and defaults to `en-us`.

Translation in the templates is done using the `trans` template tag, and in python code by the _() method. Translations of these strings are stored in message files, which contain string identifiers (typically just the original string in the default application language) and that strings translation into the target language.

To create a new messages file for language `<lang>` use the command

```
django-admin makemessages -l <lang>
```

This will extract translatable strings from the code (strings in a `trans` tag or `_()` method) and write them to a message file:

`locale/<lang>/LC_MESSAGES/django.po`

See the Django translation framework documentation for much more information about the translation framework.

See *Translation* for more details about translation implementation in Streamwebs.

Data Import

On initial deployment, the application will be seeded with data exported from the old Drupal application. A number of scripts in the `data_scripts` directory are responsible for importing data. These scripts will be run by the deployment script, and should only be run once per application instance. Due to the complexity of the Drupal exported data, modifying this code is not recommended.

User accounts will be imported from the previous application, and when the production instance is ready, every active member will be sent an email explaining how to reset their password for the new system.

See *Data Import* for more details.

Dev Environment

The Streamwebs project uses Docker and docker-compose for running test and a local instances of the application for development purposes.

The configuration in `docker-compose.yml` will build a postgres database container with the necessary PostGIS extensions, and a 'web' container running the application.

Standard docker commands can be used to run the test suite or other management commands in the web container.

See the contents of `dockerfiles/` for the Docker container definition and startup/cleanup scripts.

Setup and configuration

First, make sure you have a working Docker install, the Docker daemon is running, and your user has permission to run Docker.

Second, make sure you have the python package docker-compose installed. The easiest way to do this is to create a local python virtualenv and install docker-compose into that.

```
virtualenv venv
source venv/bin/activate
pip install docker-compose
```

Finally, configure the application. Streamwebs ships with some default settings in several 'dist' files. These files need to be copied to their proper names before the application will run.

These files may be edited, but should be adequate as-is for the Docker dev environment.

The main application settings:

```
cp streamwebs_frontend/streamwebs_frontend/settings.py.dist \
/streamwebs_frontend/streamwebs_frontend/settings.py
```

The docker environment:

```
cp dockerfiles/Dockerfile.env.dist dockerfiles/Dockerfile.env
```

Running the Dev Docker Environment

When `docker-compose` is installed and setttings are in place, you can launch the dev environment with:

```
docker-compose up web
```

This will build the PostGis container and a new CentOS container with the streamwebs application running on port 8000. You should see the application at `http://localhost:8000`.

Running management commands

To run management command in the docker environment, or to import data, you can run a bash shell in the container:

```
docker-compose run web bash
```

Migrating the database

The database schema is managed by migration files in `streamwebs_frontend/streamwebs/migrations`.

If this is the first time you've built the docker PostGis container, you will need to migrate the fresh database with the initial migration:

```
docker-compose run web bash
cd streamwebs_frontend/streamwebs_frontend
./manage.py migrate
```

Any changes to the schema should be expressed in a new migration. These can be written manually, but it is much easier and safer to let Django generate them based on changes to the models code.

After making changes to models that effect the schema, make new migrations:

```
docker-compose run web bash
cd streamwebs_frontend
./manage.py makemigrations
```

And migrate to update the schema:

```
docker-compose run web bash
cd streamwebs_frontend
./manage.py migrate
```

Importing Data

Data import scripts are idempotent, they load their data from CSV files in the `csvs` directory. The `get_all.sh` shell script will run all of the python data import scripts in the correct order. The database should be migrated prior to importing data.

```
docker-compose run web bash
cd data_scripts
./get_all
```

You can also import individual data types using the python scripts in this directory.

Creating a superuser

To create a superuser (site admin account with all permissions):

```
docker-compose run web bash
cd streamwebs_frontend
./manage.py createsuperuser
```

The script will ask for a name, password and email address.

Run a database shell

Django provides a convenient console to the database, saving you the time of manually connecting:

```
docker-compose run web bash
cd streamwebs_frontend
./manage.py dbshell
```

This will place you at a the psql commandline for the streamwebs Postgres database running in the PostGis container. Type `help` for a listing of psql commands, or `\d` to see Streamwebs' tables.

3.2 URLs

```
from django.conf.urls import include, url
import views

app_name = 'streamwebs'
urlpatterns = [
    url(r'^i18n/', include('django.conf.urls.i18n')),
    url(r'^$', views.index, name='index'),
    url(r'^faq/$', views.faq, name='faq'),
    url(r'^about/$', views.about, name='about'),


    url(r'^sites/new/$', views.create_site, name='create_site'),

    url(r'^sites/$', views.sites, name='sites'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/$', views.site, name='site'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/edit/', views.update_site,
        name='update_site'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/delete/', views.deactivate_site,
        name='deactivate_site'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/water/$',
        views.graph_water, name='graph_water'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/water/data/$',
        views.water_graph_site_data, name='water_graph_site_data'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/water/(?P<data_type>[a-zA-Z-_]+)'
        r'/(?P<date>\d{4}-\d{2}-\d{2})/',
        views.water_histogram, name='water_quality_histogram'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/macros/$',
        views.graph_macros, name='graph_macros'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/water/(?P<data_id>\d+)/',
```

```
        views.water_quality, name='water_quality'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/water/edit/',
        views.water_quality_edit, name='water_quality_edit'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/water/export/$',
        views.export_wq, name='export_wq'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/macro/(?P<data_id>\d+)/$',
        views.macroinvertebrate_view, name='macroinvertebrate_view'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/macro/edit/$',
        views.macroinvertebrate_edit, name='macroinvertebrate_edit'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/macro/export/$',
        views.export_macros, name='export_macros'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/rip_aqua/edit/$',
        views.riparian_aquatic_edit, name='rip_aqua_edit'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/rip_aqua/(?P<data_id>\d+)/$',
        views.riparian_aquatic_view, name='rip_aqua_view'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/rip_aqua/export/$',
        views.export_rip_aqua, name='export_rip_aqua'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/transect/(?P<data_id>\d+)/',
        views.riparian_transect_view, name='riparian_transect'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/transect/edit/',
        views.riparian_transect_edit, name='riparian_transect_edit'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/transect/export/$',
        views.export_ript, name='export_transects'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/canopy/(?P<data_id>\d+)/',
        views.canopy_cover_view, name='canopy_cover'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/canopy/edit',
        views.canopy_cover_edit, name='canopy_cover_edit'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/canopy/export/',
        views.export_cc, name='export_cc'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/camera/(?P<cp_id>\d+)/photo/' +
        '(?P<pp_id>\d+)/?$', views.view_pp_and_add_img, name='photo_point'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/camera/(?P<cp_id>\d+)/photo/' +
        'edit/?$', views.add_photo_point, name='photo_point_add'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/camera/(?P<cp_id>\d+)/$',
        views.camera_point_view, name='camera_point'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/camera/edit/',
        views.add_camera_point, name='camera_point_add'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/camera/$',
        views.site_camera, name='site_camera'),
```

```
    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/soil/(?P<data_id>\d+)/$',
        views.soil_survey, name='soil'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/soil/edit',
        views.soil_survey_edit, name='soil_edit'),

    url(r'^sites/(?P<site_slug>[0-9a-zA-Z-]+)/soil/export/$',
        views.export_soil, name='export_soil'),

    url(r'^statistics/$', views.admin_site_statistics, name='stats'),

    url(r'^register/$', views.register, name='register'),

    url(r'^new_org_request/(?P<school_id>[0-9]+)/$', views.new_org_request,
        name='new_org_request'),

    url(r'^register/confirm', views.confirm_registration,
        name='confirm_registration'),
    url(r'^account/$', views.account, name='account'),
    url(r'^account/update_email/$', views.update_email, name='update_email'),
    url(r'^account/update_password/$', views.update_password,
        name='update_password'),
    url(r'^login/$', views.user_login, name='login'),
    url(r'^logout/$', views.user_logout, name='logout'),

    url(r'^resources/$', views.resources, name='resources'),
    url(r'^resources/data-sheets/', views.resources_data_sheets,
        name='resources-data-sheets'),
    url(r'^resources/curriculum-guides/', views.resources_publications,
        name='resources-publications'),
    url(r'^resources/tutorial-videos/', views.resources_tutorial_videos,
        name='resources-tutorial_videos'),
    url(r'^resources/new/', views.resources_upload, name='resources-upload'),

    url(r'^schools/$', views.schools, name='schools'),
    url(r'^schools/(?P<school_id>[0-9]+)/$',
        views.school_detail, name='school_detail'),
    url(r'^get_manage_accounts/(?P<user_id>[0-9]+)$',
        views.get_manage_accounts, name='get_manage_accounts'),
    url(r'^schools/(?P<school_id>[0-9]+)/manage_accounts/$',
        views.manage_accounts, name='manage_accounts'),
    url(r'^schools/(?P<school_id>[0-9]+)/add_account/$',
        views.add_account, name='add_account'),
    url(r'^schools/(?P<school_id>[0-9]+)/edit_account/(?P<user_id>[0-9]+)/$',
        views.edit_account, name='edit_account'),

    # This is used to view a backend variable
    url(r'^var_debug/(?P<value>.*)/$',
        views.var_debug, name='var_debug')
]
```

3.3 Sites

Sites consist of a name, description, and location. A photo of the physical location is optional, a default image will be provided if no new images is uploaded.

In the previous application, Sites were called Projects, but the functionality is essentially the same.

Data is directly associated with a site, all datasheets must have a one-to-one associate with a Site.

3.3.1 The Site Model

We use Model Managers on most streamwebs models. See Django Model Managers documentation for information on why and how to use managers.

The Site model manager

```
class SiteManager(models.Manager):
    """
    Manager for the site class - creates a site to be used in tests
    """
    default = 'POINT(-121.3846841 44.0612385)'

    def create_site(self, site_name, location=default,
                    description='', image=None, active=True):

        site = self.create(site_name=site_name, location=location,
                           description=description, image=image, active=active)
        return site
```

The Site model

```
@python_2_unicode_compatible
class Site(models.Model):
    """
    The Sites model holds the information of a site, including the geographic
    location as a pair of latitudinal/longitudinal coordinates and an optional
    text description of entry.
    """

    site_name = models.CharField(max_length=250, verbose_name=_('site name'))
    description = models.TextField(blank=True,
                                   verbose_name=_('site description'))
    site_slug = models.SlugField(unique=True, max_length=50, editable=False)

    # Geo Django fields to store a point
    location = models.PointField(default='POINT(-121.3846841 44.0612385)',
                                 verbose_name=_('location'),
                                 validators=[validate_Site_location])
    image = models.ImageField(null=True, blank=True, verbose_name=_('image'),
                              upload_to='site_photos/')
    active = models.BooleanField(default=True)

    created = models.DateTimeField(default=timezone.now)
    modified = models.DateTimeField(default=timezone.now)

    objects = models.Manager()  # default manager
    test_objects = SiteManager()  # custom manager for use in writing tests
```

```
    def __str__(self):
        return self.site_name

    def to_dict(self):
        return {
            'site_name': self.site_name,
            'site_slug': self.site_slug,
            'description': self.description,
            'location': {
                'x': self.location.x,
                'y': self.location.y
            },
            'created': _timestamp(self.created),
            'modified': _timestamp(self.modified)
        }

    def save(self, **kwargs):
        'Ensure site_slug is a unique, not-null field'
        if not self.site_slug:
            self.site_slug = origSlug = slugify(self.site_name)[:50]
            # If the generated slug is not unique, append a number
            for i in xrange(1, 99):
                if not Site.objects.filter(site_slug=self.site_slug).exists():
                    break
                # Ensure the slug is never longer than it's field's maxiumum
                self.site_slug = "%s%d" % (origSlug[:50-len(str(i))], i)
        super(Site, self).save()
```

3.3.2 The Site Form

Sites can be added or edited using the Site form.

The Site form

```
class SiteForm(forms.ModelForm):
    class Meta:
        model = Site
        widgets = {
            'site_name': forms.TextInput(
                attrs={'class': 'materialize-textarea, validate'}),
            'description': forms.Textarea(
                attrs={'class': 'materialize-textarea'})
        }
        fields = ('site_name', 'description', 'location', 'image')
```

3.3.3 The Site Views

There are several views relating to Sites. There are the standard views for View, List, Create, Update, and Deactivate. Site views always include a map object displaying the Site location.

sites

In the sites view (list all sites), all available sites are presented as markers on Google map of Oregon. This view also allows searching by name, selecting a site from a pull-down list, and creating a new site. Any registered user may create a new site.

site

In the site view (view an individual site), the site location is displayed on a map along with the photo of the site and all data associated with the site. Data is displayed as a table of datasheets organized by the school which submitted the data.

Additionally, if graphable data is associated with the site (Water Quality, Macroinvertabrates, or Riparian Transect), buttons will appear to view graphs of the sumbitted data.

Finally, links are provided to add data or export data to a CSV file.

update_site

This view renders the Site form and allows the user to change the name, location, image or description.

deactivate_site

The deactivate site view will 'delete' the site by setting its `active` field to False. This can only be done if no data is associated with this site.

Site Views

```
        if UserProfile.objects.filter(user=request.user).exists():
            user_profile = UserProfile.objects.get(user=request.user)
            if user_profile is None or user_profile.school is None:
                return HttpResponseForbidden(
                    'Your account is not associated with any school.')
            return func(request, *args, **kwargs)
        else:
            return HttpResponseForbidden(
                'Your account is not associated with any school.')
    return wrapper


# Decorator function that requires the user to be a part of the
# same school as the page they are attempting to access.
def organization_required(func):
    def wrapper(request, *args, **kwargs):
        school_data = School.objects.get(id=kwargs['school_id'])

        if not request.user.has_perm('streamwebs.is_super_admin'):
            user_profile = UserProfile.objects.get(user=request.user)
            if user_profile is None:
                return HttpResponseForbidden(
                    'Your account is not associated with any school.')
            if user_profile.school != school_data:
                return HttpResponseForbidden(
                    'Your account is not associated with this school.')
        return func(request, *args, **kwargs)
    return wrapper
```

```
# Decorator function that requires the school to be active
def organization_approved(func):
    def wrapper(request, *args, **kwargs):
        school_data = School.objects.get(id=kwargs['school_id'])

        if not school_data.active:
            return HttpResponseRedirect('/schools/%i/' % school_data.id)
        return func(request, *args, **kwargs)
    return wrapper


# Send an email
def send_email(request, subject, template, user, school, from_email,
               recipients):
    send_mail(
        subject=subject,
        message='',
        html_message=render_to_string(
            template,
            {
                'protocol': request.scheme,
                'domain': request.get_host(),
                'user': user,
                'school': school
            }),
        from_email=from_email,
        recipient_list=recipients,
        fail_silently=False,
    )


def toDateTime(date, time, period):
    date_time = datetime.datetime.strptime((date + " " + time + " " + period),
                                           '%Y-%m-%d %I:%M %p')
    return date_time


def _timestamp(dt):
    return (dt - datetime.datetime(1970, 1, 1)).total_seconds()


def index(request):
    return render(request, 'streamwebs/index.html', {})


def about(request):
    return render(request, 'streamwebs/about.html', {})


def faq(request):
    return render(request, 'streamwebs/faq.html', {})


def confirm_registration(request):
    return render(request, 'streamwebs/confirm_register.html', {})
```

```
@login_required
@permission_required('streamwebs.is_org_admin', raise_exception=True)
def create_site(request):
    created = False
    site_list = Site.objects.filter(active=True)

    if request.method == 'POST':
        if not request.POST._mutable:
            request.POST._mutable = True

        if 'lat' in request.POST and 'lng' in request.POST:
            # convert lat/lng to pointfield object
            point = ("SRID=4326;POINT(%s %s)" %
                     (request.POST['lng'], request.POST['lat']))
            request.POST['location'] = point

        site_form = SiteForm(request.POST, request.FILES)
        if site_form.is_valid():
            site = site_form.save()
            site.save()
            created = True
            messages.success(request,
                             _('You have successfully added a new site.'))
            return redirect(reverse('streamwebs:site',
                            kwargs={'site_slug': site.site_slug}))

    else:
        site_form = SiteForm()

    return render(request, 'streamwebs/create_site.html', {
        'site_form': site_form,
        'created': created,
        'sites': site_list,
        'maps_api': settings.GOOGLE_MAPS_API,
        'map_type': settings.GOOGLE_MAPS_TYPE
        })


def sites(request):
    """ View for streamwebs/sites """
    site_list = Site.objects.filter(active=True).order_by('site_name')
    return render(request, 'streamwebs/sites.html', {
        'sites': site_list,
        'maps_api': settings.GOOGLE_MAPS_API,
        'map_type': settings.GOOGLE_MAPS_TYPE
    })


# view-view for individual specified site
def site(request, site_slug):
    """ View an individual site """
    site = Site.objects.filter(active=True).get(site_slug=site_slug)
    wq_sheets = Water_Quality.objects.filter(site_id=site.id)
    wq_sheets = list(wq_sheets.order_by('-date_time').values())
    wq_sheets_new = []
    for x in wq_sheets:
        wq_data = {'id': x['id'], 'uri': 'water', 'type': 'Water Quality',
```

```
                'date': x['date_time'].date()}
        if 'school_id' in x and x['school_id']:
            wq_data['school_id'] = x['school_id']
        else:
            wq_data['school_id'] = -1
```

3.4 Datasheets

3.5 Graphing

3.6 Users

3.7 Resources

3.8 Statistics

3.9 Schools

3.10 Testing

3.11 Translations

3.12 Importing Data

3.13 Miscellaneous Notes

3.13.1 Resolving the "relation does not exist" error

If you encounter the following error while running a variation of the above command or while attempting to run the application in a browser,

```
django.db.utils.ProgrammingError: relation "streamwebs_site" does not exist
LINE 1: ...ite"."created", "streamwebs_site"."modified" FROM "streamweb...
```

you probably have an unapplied migration from changes you made to a model yourself, or changes someone else made to a model that got carried over from a recent merge. To apply the migration and fix the error, run:

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

Then you can try to run your command again. If you encounter this prompt while testing,

```
Got an error creating the test database: database "test_streamwebs" already exists
Type 'yes' if you would like to try deleting the test database 'test_streamwebs', or
↪'no' to cancel:
```

go ahead and type "yes". Your tests should then be able to run. If they're not, or you're trying to run the application and the original error wasn't resolved, change directories into streamwebs_frontend/streamwebs/migrations and delete all the migrations you find there (files that begin with a number and an underscore). Afterwards you may have to run the makemigrations and migrate commands again to generate an up-to-date migration that accurately represents the current state of your models, but this should fix the problem. See the next section for more details on how this project manages migrations.

Translation

In order to make a Django project translatable, you have to add a minimal number of hooks to your Python code and templates. Django then provides utilities to extract the translation strings into a message file. This file is a convenient way for translators to provide the equivalent of the translation strings in the target language. Once the translators have filled in the message file, it must be compiled. Once this is done, Django takes care of translating Web apps on the fly in each available language, according to users' language preferences. To enable i18n we need to make the following changes in the settings.py:

```
from django.utils.translation import ugettext_lazy as _
USE_I18N = True
TEMPLATES = [
    {
        ...
        'OPTIONS': {
            'context_processors': [
                ...
                'django.template.context_processors.i18n',
            ],
        },
    },
]
```

```
MIDDLEWARE_CLASSES = (
    ...
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.common.CommonMiddleware',
    ...
)
```

Note: the order is important!!!

Specify the languages you want to use:

```
LANGUAGES = (
    ('en', 'English'),
    ('es', 'Español'),
)
```

The ugettext_lazy function is used to mark the language names for translation, and it's usual to use the function's shortcut _. Note: there is another function, ugettext, used for translation. The difference between these two functions is that ugettext translates the string immediately whereas ugettext_lazy translates the string when rendering a template. All .py files containing text for translation should have

```
from django.utils.translation import ugettext_lazy as _
```

towards the top of the file. Also settings.py should have local_paths specified

```
LOCALE_PATHS = (
    '../locale/',
)
```

The urls.py should contain this:

```
url(r'^i18n/', include('django.conf.urls.i18n'))
```

Finally, mark the text you want to translate by wrapping it into _(‘ ‘), i.e.: _(‘Password’).

Template files have to contain

```
{% load i18n %}
```

at the top and the text to be translated has to be wrapped around {% ‘ ‘ %}, i.e.:

```
{% 'Username:' %}
```

The string for translating Streamwebs website are extracted into locale/.po files. In docker run

```
python manage.py makemessages -l <language code>
```

(i.e., ‘python manage.py makemessages -l es’ for Spanish). If this is a new language just added into the settings.py, the command will create a new directory in the locale folder with the .po file. If the language already existed, the command will update the .po file.

After translating all the strings in .po file run

```
python manage.py compilemessages
```

This runs over all available .po files and creates .mo files, which are binary files optimized for use by gettext. For translators: use .po file of your working language, complete the space in the empty parenthesis with the translations:

```
#: streamwebs/templates/streamwebs/register.html:11
 msgid "Create an account."
 msgstr " "
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search