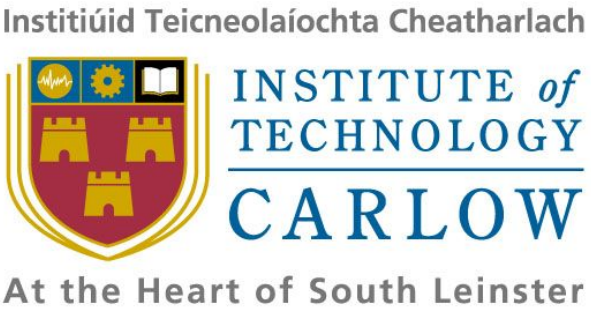

Online Voting System

Code Document

Name: Mark Kelly

Student Number: C00198041

Course: Bachelor of Science (Honours) Software Development

Supervisor: Dr. Lei Shi

Date: 18th April 2018

Table of Contents

1. Requirements

```
Django==1.9
django-admin-bootstrapped==2.5.7
django-authtools==1.6.0
django-autocomplete-light==3.3.0rc5
django-braces==1.12.0
django-crispy-forms==1.7.0
django-debug-toolbar==1.9.1
django-environ==0.4.4
easy-thumbnails==2.5
mysqlclient==1.3.12
Pillow==5.0.0
python-http-client==3.0.0
sendgrid==3.6.5
sendgrid-django==4.2.0
six==1.11.0
sqlparse==0.2.4
Werkzeug==0.14.1
```

2. Src

This directory contains all the applications for the project.

2.1 manage.py

```
#!/usr/bin/env python
import os
import sys

if __name__ == "__main__":
    # CHANGED manage.py will use development settings by
    # default. Change the DJANGO_SETTINGS_MODULE environment variable
```

```
    # for using the environment specific settings file.
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "evoteproject.settings.development")

    from django.core.management import execute_from_command_line

    execute_from_command_line(sys.argv)
```

3. Accounts

The accounts app handles logging in, signing up, and password changes and resets.

3.1 forms.py

```
from __future__ import unicode_literals
from django.contrib.auth.forms import AuthenticationForm
from django import forms
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Layout, Div, Submit, HTML, Button, Row, Field, Hidden
from crispy_forms.bootstrap import AppendedText, PrependedText, FormActions
from authtools import forms as authtoolsforms
from django.contrib.auth import forms as authforms
from django.core.urlresolvers import reverse

class LoginForm(AuthenticationForm):
    remember_me = forms.BooleanField(required=False, initial=False)

    def __init__(self, *args, **kwargs):
        super(LoginForm, self).__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.fields["username"].widget.input_type = "email"  # ugly hack

        self.helper.layout = Layout(
            Field('username', placeholder="Enter Email", autofocus=""),
            Field('password', placeholder="Enter Password"),
            HTML('<a href="{}">Forgot Password?</a>'.format(
                reverse("accounts:password-reset"))),
            Field('remember_me'),
            Submit('sign_in', 'Log in',
                   css_class="btn btn-lg btn-success btn-block"),
            )


class SignupForm(authtoolsforms.UserCreationForm):

    def __init__(self, *args, **kwargs):
        super(SignupForm, self).__init__(*args, **kwargs)
        self.helper = FormHelper()
        self.fields["email"].widget.input_type = "email"  # ugly hack

        self.helper.layout = Layout(
            Field('email', placeholder="Enter Email", autofocus=""),
            Hidden('name', placeholder="Enter Full Name", value="Voter"),
```

```
            Field('password1', placeholder="Enter Password"),
            Field('password2', placeholder="Re-enter Password"),
            Submit('sign_up', 'Register', css_class="btn btn-lg btn-success btn-block"),
            )


class PasswordChangeForm(authforms.PasswordChangeForm):

    def __init__(self, *args, **kwargs):
        super(PasswordChangeForm, self).__init__(*args, **kwargs)
        self.helper = FormHelper()

        self.helper.layout = Layout(
            Field('old_password', placeholder="Enter old password",
                  autofocus=""),
            Field('new_password1', placeholder="Enter new password"),
            Field('new_password2', placeholder="Enter new password (again)"),
            Submit('pass_change', 'Change Password', css_class="btn-success"),
            )


class PasswordResetForm(authtoolsforms.FriendlyPasswordResetForm):

    def __init__(self, *args, **kwargs):
        super(PasswordResetForm, self).__init__(*args, **kwargs)
        self.helper = FormHelper()

        self.helper.layout = Layout(
            Field('email', placeholder="Enter email",
                  autofocus=""),
            Submit('pass_reset', 'Reset Password', css_class="btn-success"),
            )


class SetPasswordForm(authforms.SetPasswordForm):
    def __init__(self, *args, **kwargs):
        super(SetPasswordForm, self).__init__(*args, **kwargs)
        self.helper = FormHelper()

        self.helper.layout = Layout(
            Field('new_password1', placeholder="Enter new password",
                  autofocus=""),
            Field('new_password2', placeholder="Enter new password (again)"),
            Submit('pass_change', 'Change Password', css_class="btn-success"),
            )
```

3.2 tokens.py

```
from django.contrib.auth.tokens import PasswordResetTokenGenerator
from django.utils import six

class TokenGenerator(PasswordResetTokenGenerator):
    def _make_hash_value(self, user, timestamp):
        return (
```

```
            six.text_type(user.pk) + six.text_type(timestamp) +
            six.text_type(user.is_active)
        )
account_activation_token = TokenGenerator()
```

3.3 urls.py

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

urlpatterns = [
    url(r'^login/$', views.LoginView.as_view(), name="login"),
    url(r'^logout/$', views.LogoutView.as_view(), name='logout'),
    url(r'^register/$', views.SignUpView.as_view(), name='signup'),
    url(r'^password-change/$', views.PasswordChangeView.as_view(),
        name='password-change'),
    url(r'^password-reset/$', views.PasswordResetView.as_view(),
        name='password-reset'),
    url(r'^password-reset-done/$', views.PasswordResetDoneView.as_view(),
        name='password-reset-done'),

url(r'^password-reset/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1
,20})/$$', views.PasswordResetConfirmView.as_view(),  # NOQA
        name='password-reset-confirm'),

url('^activate/(?P<uidb64>[0-9A-Za-z_\-]+)/(?P<token>[0-9A-Za-z]{1,13}-[0-9A-Za-z]{1,20})/$
',
        views.activate, name='activate'),
]
```

3.4 views.py

```
from __future__ import unicode_literals
from django.core.urlresolvers import reverse_lazy
from django.views import generic
from django.contrib.auth import get_user_model
from django.contrib import auth
from django.contrib import messages
from authtools import views as authviews
from braces import views as bracesviews
from django.conf import settings
from django.shortcuts import render, redirect
from . import forms
from django.contrib.auth.password_validation import validate_password
from django.core.exceptions import ValidationError
from django_elect.models import Election, Vote
import pandas as pd
import os
from django.http import HttpResponse
from django.contrib.auth import login, authenticate
from .forms import SignupForm
from django.contrib.sites.shortcuts import get_current_site
from django.utils.encoding import force_bytes, force_text
```

```
from django.utils.http import urlsafe_base64_encode, urlsafe_base64_decode
from django.template.loader import render_to_string
from .tokens import account_activation_token
from django.core.mail import EmailMessage

User = get_user_model()


class LoginView(bracesviews.AnonymousRequiredMixin,
                authviews.LoginView):
    template_name = "accounts/login.html"
    form_class = forms.LoginForm

    def form_valid(self, form):
        redirect = super(LoginView, self).form_valid(form)
        remember_me = form.cleaned_data.get('remember_me')
        # messages.success(self.request, "Login successful")
        if remember_me is True:
            ONE_MONTH = 30*24*60*60
            expiry = getattr(settings, "KEEP_LOGGED_DURATION", ONE_MONTH)
            self.request.session.set_expiry(expiry)
        return redirect


class LogoutView(authviews.LogoutView):
    url = reverse_lazy('home')


def activate(request, uidb64, token):
    try:
        uid = force_text(urlsafe_base64_decode(uidb64))
        user = User.objects.get(pk=uid)
    except (TypeError, ValueError, OverflowError, User.DoesNotExist):
        user = None

    if user is not None and account_activation_token.check_token(user, token):
        user.is_active = True
        # user.profile.email_confirmed = True
        user.save()
        messages.success(request, "Your email has been confirmed, "
                         "you may now login")
        # auth.login(request, user)
        return redirect('/')
    else:
        return render(request, 'accounts/account-activation-invalid.html')


class SignUpView(bracesviews.AnonymousRequiredMixin,
                 bracesviews.FormValidMessageMixin,
                 generic.CreateView):
    form_class = forms.SignupForm
    model = User
    template_name = 'accounts/signup.html'
    success_url = reverse_lazy('accounts:logout')
```

```
    form_valid_message = "Registration successful"

    def form_valid(self, form):
        election = Election.get_latest_or_404()
        df = pd.read_excel(election.voter_register.path, sheetname=0)

        def check_student_register(email):
            for index, row in df.iterrows():
                if row['Email'] == email:
                    return True
            return False

        if form.is_valid():
            user = form.save(commit=False)  # Do not save to table yet
            user.is_active = False
            username = form.cleaned_data['email']
            password = form.cleaned_data['password1']

            if check_student_register(username) is False:
                messages.warning(self.request, "This email address is not allowed to vote
in this election")
                return render(self.request, self.template_name, {'form': form})

            try:
                validate_password(password, user)
            except ValidationError as e:
                form.add_error('password1', e)  # to be displayed with the field's errors
                return render(self.request, self.template_name, {'form': form})

            user.save()
            current_site = get_current_site(self.request)
            mail_subject = 'Activate your voting account.'
            message = render_to_string('accounts/emails/account-activation-email.html', {
                'user': user,
                'domain': current_site.domain,
                'uid':urlsafe_base64_encode(force_bytes(user.pk)),
                'token':account_activation_token.make_token(user),
            })
            to_email = form.cleaned_data.get('email')
            email = EmailMessage(
                        mail_subject, message, to=[to_email]
            )
            email.send()
            return render(self.request, 'accounts/account-activation-sent.html')

        r = super(SignUpView, self).form_valid(form)
        username = form.cleaned_data["email"]
        password = form.cleaned_data["password1"]
        user = form.save()
        user = auth.authenticate(email=username, password=password)
        auth.login(self.request, user)
        return r
```

```
class PasswordChangeView(authviews.PasswordChangeView):
    form_class = forms.PasswordChangeForm
    template_name = 'accounts/password-change.html'
    success_url = reverse_lazy('home')

    def form_valid(self, form):
        form.save()
        messages.success(self.request,
                         "Your password was changed, "
                         "hence you have been logged out. Please relogin")
        return super(PasswordChangeView, self).form_valid(form)


class PasswordResetView(authviews.PasswordResetView):
    form_class = forms.PasswordResetForm
    template_name = 'accounts/password-reset.html'
    success_url = reverse_lazy('accounts:password-reset-done')
    subject_template_name = 'accounts/emails/password-reset-subject.txt'
    email_template_name = 'accounts/emails/password-reset-email.html'


class PasswordResetDoneView(authviews.PasswordResetDoneView):
    template_name = 'accounts/password-reset-done.html'


class PasswordResetConfirmView(authviews.PasswordResetConfirmAndLoginView):
    template_name = 'accounts/password-reset-confirm.html'
    form_class = forms.SetPasswordForm
```

3.5 Accounts templates

3.5.1 account-activation-email.html

```
{% load i18n %}{% autoescape off %}
Hi {{ user.get_username }},
Please click on the link below to confirm your registration:
{% block reset_link %}
http://www.{{ domain }}{% url 'accounts:activate' uidb64=uid token=token %}
{% endblock %}
{% endautoescape %}
```

3.5.2 password-reset-email.html

```
{% load i18n %}{% autoescape off %}
{% blocktrans %}You're receiving this email because you requested a password reset for your
user account at {{ site_name }}.{% endblocktrans %}
{% trans "Please go to the following page and choose a new password:" %}
{% block reset_link %}
{{ protocol }}://{{ domain }}{% url 'accounts:password-reset-confirm' uidb64=uid
token=token %}
{% endblock %}
{% trans "Your login email, in case you've forgotten, is same as this email address:" %} {{
user.get_username }}
```

```
{% trans "Thanks for using our site!" %}
{% blocktrans %}The {{ site_name }} team{% endblocktrans %}
{% endautoescape %}
```

3.5.3 account-activation-invalid.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Invalid{% endblock %}
{% block form_heading %}Invalid{% endblock %}
{% block form %}
<div class="form-message">
  <p>
    The confirmation link was invalid, possibly because it has already been used.
  </p>
</div>
{% endblock form %}
```

3.5.4 account-activation-sent.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Confirm Email Sent{% endblock %}
{% block form_heading %}Confirm Email Sent{% endblock %}
{% block form %}
<div class="form-message">
  <p>
    Please check your mail. You would have received a confirmation email from us.
  </p>
</div>
{% endblock form %}
```

3.5.5 auth_base.html

```
{% extends "base.html" %}
{% load crispy_forms_tags %}
{% block navbar %}
{% endblock %}
{% block container %}
<div class="container form-box">
  <div class="text-center">
    <h2>{% block form_heading %}{% endblock %}</h2>
  </div>
  {% block form %}
  {% endblock %}
</div>
{% endblock container %}
```

3.5.6 login.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Log In{% endblock %}
{% block form_heading %}Please Log In{% endblock %}
```

```
{% block form %}
  {% crispy form %}
  <div class="form-message">
    <p>
      Not registered to vote? <a href="{% url 'accounts:signup' %}">Register</a>.
    </p>
  </div>
{% endblock form %}
```

3.5.7 password-change.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Password Change{% endblock %}
{% block form_heading %}Password Change{% endblock %}
{% block form %}
  {% crispy form %}
{% endblock form %}
```

3.5.8 password-reset.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Password Reset{% endblock %}
{% block form_heading %}Password Reset{% endblock %}
{% block form %}
  {% crispy form %}
{% endblock form %}
```

3.5.9 password-reset-confirm.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Password Change{% endblock %}
{% block form_heading %}Password Change{% endblock %}
{% block form %}
  {% crispy form %}
{% endblock form %}
```

3.5.10 password-reset-done.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Password Reset Done{% endblock %}
{% block form_heading %}Password Reset Email Sent{% endblock %}
{% block form %}
<div class="form-message">
  <p>
    Please check your mail. You would have received a password reset mail from us.
  </p>
</div>
{% endblock form %}
```

3.5.11 signup.html

```
{% extends "accounts/auth_base.html" %}
{% load crispy_forms_tags %}
{% block title %}{{ block.super }}Register{% endblock %}
{% block form_heading %}Register to Vote{% endblock %}
{% block form %}
  {% crispy form %}
  <div class="form-message">
    <p>
      Already registered to vote? <a href="{% url 'accounts:login' %}">Log in</a>.
    </p>
  </div>
{% endblock form %}
```

4. django_elect

This application handles the elections and voting.

4.1 admin.py

```
from django.core.urlresolvers import reverse
from django.contrib import admin
from django.http import HttpResponseRedirect
from django import forms
from dal import autocomplete
from django_elect.models import Election, Ballot, Vote, Candidate, \
    VotePreferential, VotePlurality
from django_elect import settings

class BallotInline(admin.StackedInline):
    model = Ballot
    extra = 3


class ElectionAdmin(admin.ModelAdmin):
    actions_html = """
        <a href="%s">View Statistics</a> |
        <a href="%s">Generate Excel Spreadsheet</a> |
        <a href="%s/">Disassociate Accounts</a>
    """
    list_display = ('name', 'vote_start', 'vote_end', 'admin_actions')
    filter_horizontal = ("allowed_voters",)
    inlines = [BallotInline]
    exclude = ('allowed_voters',)

    def admin_actions(self, obj):
        kwargs = {'id': str(obj.pk)}
        return self.actions_html % (
            reverse('voting:django_elect_stats', kwargs=kwargs),
            reverse('voting:django_elect_spreadsheet', kwargs=kwargs),
```

```
            reverse('voting:django_elect_disassociate', kwargs=kwargs),
        )
    admin_actions.short_description = "Administrative Actions"
    admin_actions.allow_tags = True

    def response_add(self, request, obj):
        # Overrides ModeAdmin.response_add() and redirects user to the ballot
        # page, filtered for the new Election
        msg = "The election '%s' was added successfully. " % unicode(obj)
        msg += """Please fill in the details for all the ballots listed
           below. Use the "Add Ballot" button to add additional ballots."""
        self.message_user(request, msg)
        url = "../../ballot/?election__id__exact=%i" % obj.pk
        return HttpResponseRedirect(url)


class CandidateInline(admin.StackedInline):
    model = Candidate
    extra = 5
    exclude = ('biography', 'write_in')


class CandidateAdmin(admin.ModelAdmin):
    exclude = ('write_in', 'biography')


class BallotAdmin(admin.ModelAdmin):
    list_display = ("election", "description", "type")
    inlines = [CandidateInline]
    exclude = ('write_in_available',)


class AdminVotePreferentialForm(forms.ModelForm):
    class Meta:
        model = VotePreferential
        fields = '__all__'
        widgets = {
            'candidate': autocomplete.ModelSelect2(forward=['election'],
                url='vote-preferential-autocomplete')
        }


class VotePreferentialInline(admin.TabularInline):
    model = VotePreferential
    form = AdminVotePreferentialForm


class AdminVotePluralityForm(forms.ModelForm):
    class Meta:
        model = VotePlurality
        fields = '__all__'
        widgets = {
            'candidate': autocomplete.ModelSelect2(forward=['election'],
                url='vote-plurality-autocomplete')
```

```
        }


class VotePluralityInline(admin.TabularInline):
    model = VotePlurality
    form = AdminVotePluralityForm


class AdminVoteForm(forms.ModelForm):
    class Meta:
        model = Vote
        fields = '__all__'
        widgets = {
            'account': autocomplete.ModelSelect2(url='account-autocomplete'),
        }


class VoteAdmin(admin.ModelAdmin):
    form = AdminVoteForm
    list_display = ('election', 'account')
    list_filter = ['election']
    search_fields = ['account__first_name', 'account__last_name']
    inlines = [VotePreferentialInline, VotePluralityInline]

    class Media:
        js = ('django_elect/js/admin.js',)


admin.site.register(Candidate, CandidateAdmin)
admin.site.register(Election, ElectionAdmin)
admin.site.register(Ballot, BallotAdmin)
```

4.2 apps.py

```
from django.apps import AppConfig

class DjangoElectConfig(AppConfig):
    name = 'django_elect'
    verbose_name = "ITCSU voting"
```

4.3 autocomplete.py

```
from django.apps import apps
from django.contrib.admin.views.decorators import staff_member_required
from django.db.models.functions import Concat
from django.db.models import Value as V
from django.utils.decorators import method_decorator
from dal import autocomplete
from django_elect import settings
from django_elect.models import Candidate

class CandidateAutocomplete(autocomplete.Select2QuerySetView):
    ballot_type = None
```

```
    @method_decorator(staff_member_required)
    def dispatch(self, *args, **kwargs):
        return super(CandidateAutocomplete, self).dispatch(*args, **kwargs)

    def get_queryset(self):
        if not self.ballot_type:
            raise "Ballot type not specified"

        qs = Candidate.objects.annotate(
            full_name=Concat('first_name', V(' '), 'last_name'),
        ).filter(ballot__type=self.ballot_type)

        election = self.forwarded.get('election', None)
        if election:
            qs = qs.filter(ballot__election=election)

        if self.q:
            qs = qs.filter(full_name__icontains=self.q)

        return qs


class AccountAutocomplete(autocomplete.Select2QuerySetView):
    @method_decorator(staff_member_required)
    def dispatch(self, *args, **kwargs):
        return super(AccountAutocomplete, self).dispatch(*args, **kwargs)

    def get_queryset(self):
        user_model = apps.get_model(settings.DJANGO_ELECT_USER_MODEL)
        qs = user_model.objects.all()

        if self.q:
            qs = settings.DJANGO_ELECT_USER_AUTOCOMPLETE_FILTER(qs, self.q)

        return qs
```

4.4 forms.py

```
from string import Template
from django import forms
from django.conf import settings
from django.core.urlresolvers import reverse
from django.utils.http import urlquote
from django.utils.safestring import mark_safe
from django_elect.models import Candidate, VotePlurality, VotePreferential

class CandidateRowWidget(forms.Widget):
    """
    Form widget for showing a table row with information on a single candidate.
    """
    def __init__(self, candidate, form_widget, template, *args, **kwargs):
        """
        "candidate" is the Candidate to show, and "form_widget" is the form
```

```
        input to show next to the candidate's name
        """
        super(CandidateRowWidget, self).__init__(*args, **kwargs)
        self.candidate = candidate
        self.form_widget = form_widget
        self.template = template

    def value_from_datadict(self, data, files, name):
        return self.form_widget.value_from_datadict(data, files, name)

    def render(self, name, value, attrs=None):
        candidate_name = self.candidate.get_name()
        if self.candidate.biography:
            # make candidate's name a link to the appropriate anchor
            # on the auto-generated biographies page
            candidate_name = '<a target="_blank" href="%s/#%s">%s</a>' % (
                reverse('voting:django_elect_biographies'),
                urlquote(candidate_name),
                candidate_name,
            )
        select = self.form_widget.render(name, value, attrs)
        photo_unavailable = settings.STATIC_URL + \
                            "django_elect/img/default_profile.png"

        return mark_safe(self.template.substitute({
            'select': select,
            # 'incum': self.candidate.incumbent and "*" or "",
            'name': candidate_name,
            'inst': self.candidate.institution or "N/A",
            'image': self.candidate.picture or 'default_profile.png',
        }))

class BaseVoteForm(forms.Form):
    """
    Base class for representing the form for a single ballot
    """
    ballot = None
    candidate_list = None

    def __init__(self, ballot, *args, **kwargs):
        super(BaseVoteForm, self).__init__(*args, **kwargs)
        self.ballot = ballot

    def __unicode__(self):
        output = ['<table class="table table-hover">', self.get_table_info()['header']]
        for name, field in self.fields.items():
            bf = forms.forms.BoundField(self, field, name)
            output.append(unicode(bf))
        output.append('</table>')
        return mark_safe(u'\n'.join(output))

    def has_candidates(self):
        """
        Returns True if this form is valid and contains at least one candidate
```

```
        """
        return self.is_valid() and len(self.candidate_list) >= 1

    def get_table_info(self):
        """
        Returns the string to use for constructing the ballot table's header
        row and a Template object to use for each body row.

        This is done dynamically so that the columns for "candidate image" and
        "candiate institution" are ommitted if no candidates in the ballot have
        a image or institution defined.
        """
        candidates = self.ballot.candidates.filter()
        # candidates = self.ballot.candidates.filter(write_in=False)
        has_image = any([c.picture for c in candidates])
        has_institution = any([c.institution for c in candidates])
        header_cols = """
            <col class="ballot-col-image" />
            <col class="ballot-col-name" />
            <col class="ballot-col-select" />"""
        header = """
            <tr>
            <thead style='display:none;'>

            </thead>"""
        row_template = """
            <tr class="candidate-row">
                <td class="text-center"><img class="candidate-photo" src="/media/$image"
height="70" width="70" alt="Photograph"/></td>
                <td class="text-left"><b>$name</b></td>
                <td class="text-center">$select</td>"""
        if has_institution:
            row_template += "<td>$inst</td>"
            header_cols += '<col class="ballot-col-institution"/>'
            header += '<th>Institution</th>'
        # if has_image:
        #row_template += '<td><img class="candidate-photo" src="$image" height="30"
width="30" alt="Photograph"/></td>'
        #header_cols += '<col class="ballot-col-image" />'
        #header += '<th>Picture</th>'
        row_template += '</tr>'
        #row_template += '<p>test</p>'
        #header += '</tr>'
        return {
            'header': header_cols + header,
            'row_template': Template(row_template),
        }

    def save(self, vote):
        """
        Creates appropriate objects for each candidate choice and associates
        them with the given Vote object, unless the associated ballot
        is secret.
        """
```

```
        #if self.ballot.is_secret:
         #   vote = None
        return self._do_save(vote)

    def _do_save(self, vote):
        """
        Must be implemented by sub-classes to save VotePreferential and
        VotePlurality objects
        """
        raise NotImplementedError

    def get_write_in_candidate(self, write_in):
        """
        Returns write-in candidate corresponding to given dictionary if this
        ballot has been marked to accept write-in candidates, else returns None
        """
        if not self.ballot.write_in_available or not write_in:
            return None
        return Candidate.objects.get_or_create(ballot=self.ballot,
            write_in=True, first_name=write_in['first_name'],
            last_name=write_in['last_name'])[0]


class WriteInField(forms.MultiValueField):
    """
    Represents the write-in candidate field for a general ballot.
    Must be passed the appropriate widget.
    """
    def __init__(self, widget, fields=(), *args, **kwargs):
        fields += (
            forms.CharField(label="First Name", min_length=1, max_length=45),
            forms.CharField(label="Last Name", min_length=1, max_length=45),
        )
        self.widget = widget(widgets=[f.widget for f in fields])
        super(WriteInField, self).__init__(fields, *args, **kwargs)
        self.required = False

    def compress(self, data_list):
        if data_list:
            return {'first_name': data_list[0], 'last_name': data_list[1]}
        return None

    def clean(self, value):
        name = super(WriteInField, self).clean(value)
        if name and ((name['first_name'] and not name['last_name']) or
           (not name['first_name'] and name['last_name'])):
            message = "Please enter in both the first and last name for "+\
                      "write-in candidates."
            raise forms.ValidationError(message)
        return name


class PluralityVoteForm(BaseVoteForm):
    """
```

```
    Extends BaseVoteForm to implement the vote form for plurality ballots.
    """
    def __init__(self, *args, **kwargs):
        super(PluralityVoteForm, self).__init__(*args, **kwargs)
        template = self.get_table_info()['row_template']
        candidates = self.ballot.candidates.filter(write_in=False)
        if self.ballot.seats_available == 1 and \
            not self.ballot.write_in_available:
            select = self.RadioWidget(self.ballot.pk)
        else:
            select = forms.CheckboxInput()
        for candidate in candidates:
            widget = CandidateRowWidget(candidate, select, template)
            self.fields[candidate.pk] = forms.BooleanField(label="",
                widget=widget, required=False)
        if self.ballot.write_in_available:
            self.fields['write_in'] = WriteInField(widget=self.WriteInWidget)

    class RadioWidget(forms.widgets.Widget):
        """
        Slightly hackish widget for representing two candidates on a single
        ballot using radio buttons. Ensures the names of the radio inputs are
        the same.
        """
        def __init__(self, ballot_id):
            self.name = "ballot%i" % ballot_id
            self.attrs = {}

        def render(self, value, name=None, attrs=None):
            final_attrs = self.build_attrs(attrs, type='radio',
                value=value, name=self.name)
            return mark_safe(u'<input%s />' % forms.utils.flatatt(final_attrs))

        def value_from_datadict(self, data, files, name):
            value = data.get(unicode(self.name))
            return (value == name)

    class WriteInWidget(forms.MultiWidget):
        """
        Widget representing the "write-in" choice on a plurality ballot.
        """
        def decompress(self, value):
            if value:
                return (value['first_name'], value['last_name'])
            else:
                return (None, None, None)

        def format_output(self, rendered_widgets):
            return mark_safe(u"""
                <tr>
                    <th colspan="2"> </th>
                    <th>First Name</th>
                    <th>Last Name</th>
                </tr>
```

```
                <tr>
                    <td colspan="2">Write in:</td>
                    <td>%s</td>
                    <td>%s</td>
                </tr>""" % tuple(rendered_widgets))

    def clean(self):
        clean = super(PluralityVoteForm, self).clean()
        candidates = [cand for cand, selected in clean.items()
                      if selected and isinstance(cand, long)]
        self.candidate_list = Candidate.objects.in_bulk(candidates).values()
        seats = self.ballot.seats_available
        write_in = clean.get('write_in')
        if (len(self.candidate_list) + (write_in and 1 or 0)) > seats:
            message = 'Please select %i or fewer candidates.' % (seats)
            raise forms.ValidationError(message)

        if write_in:
            self.candidate_list.append(self.get_write_in_candidate(write_in))
        return clean

    def _do_save(self, vote):
        for candidate in self.candidate_list:
            VotePlurality(vote=vote, candidate=candidate).save()


class PreferentialVoteForm(BaseVoteForm):
    """
    Extends BaseVoteForm to implement the vote form for preferential ballots.
    """
    def __init__(self, *args, **kwargs):
        super(PreferentialVoteForm, self).__init__(*args, **kwargs)
        template = self.get_table_info()['row_template']
        candidates = self.ballot.candidates.filter()
        #we use the Borda count method for preferential ballots, so each
        #candidate select box should have options in the format
        #[(0, 0), (1, 3), (2, 2), (3, 1)]
        point_options = [(0, 0)]
        if self.ballot.write_in_available:
            points = range(1, candidates.count() + 2)
        else:
            points = range(1, candidates.count() + 1)
        point_options += zip(points[::], points) # Changed select value here ***********
        print(type(points))
        select = forms.Select(choices=point_options,
            attrs={'style': 'width: 40px'})
        for candidate in candidates:
            widget = CandidateRowWidget(candidate, select, template)
            self.fields[candidate.pk] = forms.ChoiceField(label="",
                choices=point_options, widget=widget)
        if self.ballot.write_in_available:
            self.fields['write_in'] = PreferentialWriteInField(
                choices=point_options)
```

```
    def clean(self):
        clean = super(PreferentialVoteForm, self).clean()
        write_in = clean.pop('write_in', False)
        point_list = [int(i) for i in clean.values() if int(i) > 0]
        if write_in and write_in['points'] > 0:
            point_list.append(write_in['points'])
        message = ""
        # check that no point value exceeds the number of candidates
        candidates = self.ballot.candidates.filter()
        num = candidates.count()
        if point_list and filter(lambda x: x > num, point_list):
            message = "Please rank your preferences from 1 to %i." % num
        # check that there are no duplicate points
        elif len(set(point_list)) != len(point_list) and sum(point_list) > 0:
            message = "Please do not give the same point value (other than "+\
                      "zero) to more than one candidate."
        if message:
            raise forms.ValidationError(message)
        self.candidate_list = [(candidates.get(id=c), int(p))
                               for c, p in clean.items() if int(p) > 0]
        if write_in:
            candidate = self.get_write_in_candidate(write_in)
            self.candidate_list.append((candidate, write_in['points']))
        return clean

    def _do_save(self, vote):
        for candidate, points in self.candidate_list:
            VotePreferential(vote=vote, candidate=candidate,
                             point=points).save()


class PreferentialWriteInField(WriteInField):
    """
    WriteInField with an additional ChoiceField.
    """
    def __init__(self, choices, *args, **kwargs):
        fields = (
            forms.ChoiceField(choices=choices),
        )
        super(PreferentialWriteInField, self).__init__(
            fields=fields, widget=self.WriteInWidget, *args, **kwargs)

    def compress(self, data_list):
        # make sure something was filled in for name
        if data_list and (data_list[1] or data_list[2]):
            return {'points': int(data_list[0]),
                    'first_name': data_list[1],
                    'last_name': data_list[2]}
        return None

    def clean(self, value):
        write_in = super(PreferentialWriteInField, self).clean(value)
        if write_in and write_in['first_name'] and write_in['last_name'] and \
           (write_in['points'] == 0):
```

```
            message = "Please select a non-zero point value for the write-in"+\
                      " candidate you entered."
            raise forms.ValidationError(message)
        return write_in

    class WriteInWidget(forms.MultiWidget):
        """
        Widget representing the "write-in" choice on a preferential ballot.
        """
        def decompress(self, value):
            if value:
                return (value['points'], value['first_name'],
                        value['last_name'])
            else:
                return (None, None, None)

        def format_output(self, rendered_widgets):
            return mark_safe(u"""

                <tr class="candidate-row">
                    <td></td>
                    <td class="text-left"><b>Reopen Nominations<b></td>
                    <td>%s</td>
                    <td>%s</td>
                    <td>%s</td>
                </tr>""" % tuple(rendered_widgets))
```

4.5 models.py

```
from datetime import datetime
from django.http import Http404
from django.db import models, connection
from django.db.models import Q
from django_elect import settings
from .validators import validate_file_extension, validate_image_extension

class VotingNotAllowedException(Exception):
    pass


class Election(models.Model):
    """
    Represents elections. which can be composed of one or more ballots.
    Voting only allowed between vote_start and vote_end dates.
    """
    name = models.CharField(max_length=255, blank=False, unique=True,
        help_text="Used to uniquely identify elections. Will be shown "+\
        "with ' Election' appended to it on all publicly-visible pages.")
    introduction = models.TextField(blank=True,
        help_text="This is printed at the top of the voting page below "+\
        "the header")
    vote_start = models.DateTimeField(help_text="Start of voting")
    vote_end = models.DateTimeField(help_text="End of voting")
```

```
    voter_register = models.FileField('Allowed voters',
                                #upload_to="static/django_elect/img",
                                upload_to='student_register/%d-%m-%Y/',
                                validators=[validate_file_extension],
                                null=True,
                                blank=False,
                                help_text = "Please upload Excel file")

    allowed_voters = models.ManyToManyField(settings.DJANGO_ELECT_USER_MODEL,
        blank=True,
        help_text="If empty, all registered users will be allowed to vote.")

    def __unicode__(self):
        return unicode(self.name)

    def voting_allowed_for_user(self, user):
        """
        Returns True if not is between vote_start and vote_end, inclusive,
        and given user is in allowed_voters and user hasn't already voted.
        """
        return (self.voting_allowed() and not self.has_voted(user) and
            (self.allowed_voters.count() == 0 or
            self.allowed_voters.filter(id=user.id)))

    def voting_allowed(self):
        """
        Returns True if now is between vote_start and vote_end, inclusive.
        """
        return self.vote_start <= datetime.now() <= self.vote_end

    def create_vote(self, user):
        """
        Checks that the given account can vote in this election, and if so,
        creates and returns a Vote object.
        """
        if not self.voting_allowed_for_user(user):
            msg = 'The account %s is not allowed to vote in this election.'
            raise VotingNotAllowedException(msg % unicode(user))
        return self.votes.create(account=user, election=self)

    def has_voted(self, account):
        """ Returns True if given account has voted for this election """
        return self.votes.filter(account=account).count() != 0

    def get_full_statistics(self):
        """
        Returns dictionary of the following form:
        {
            "candidates": [ Candidate#1, Candidate#2, ... ],
            "ballots": [ Ballot#1, Ballot#2, ... ],
            "votes": {
                Vote#1: [ points_for_candidate1, points_for_candidate2, ... ],
                Vote#2: [...],
            },
```

```
}
Where "points_for_candidate#" is 0 if the vote doesn't contain the
corresponding candidate and either the point value (for preferential
ballots) or 1 (for plurality) if so. Candidates are ordered by ballot
ID and then by candidate id.
"""
query = """
    SELECT
        v.id AS vote_id,
        c.id AS candidate_id,
        IF(
            b.type = "Pl",
            IF(vpl.id IS NULL, 0, 1),
            IFNULL(SUM(vpr.point), 0)
        ) AS point
    FROM %(vote)s v
    JOIN %(ballot)s b ON (b.election_id = v.election_id)
    JOIN %(candidate)s c ON (c.ballot_id = b.id)
    LEFT JOIN %(vote_plurality)s vpl ON (vpl.vote_id = v.id
        AND vpl.candidate_id = c.id)
    LEFT JOIN %(vote_preferential)s vpr ON (vpr.vote_id = v.id
        AND vpr.candidate_id = c.id)
    WHERE v.election_id = '%(id)i'
    GROUP BY v.id, c.id
    ORDER BY v.id, b.id, c.id
"""
query %= {
    'vote': Vote._meta.db_table,
    'ballot': Ballot._meta.db_table,
    'candidate': Candidate._meta.db_table,
    'vote_plurality': VotePlurality._meta.db_table,
    'vote_preferential': VotePreferential._meta.db_table,
    'id': self.pk,
}

cursor = connection.cursor()
cursor.execute(query)

stats = {
    'candidates': [],
    'ballots': [],
    'votes': {},
}

vote_points = {}
for row in cursor.fetchall():
    vote_id, candidate_id, point = row[0], row[1], row[2]
    vote_points.setdefault(vote_id, []).append(point)
    # Every vote in the result set has the same list of candidates, so
    # we only need to populate the candidates and ballots lists on
    # the first one.
    if len(vote_points) == 1:
        candidate = Candidate.objects.get(id=candidate_id)
        stats['candidates'].append(candidate)
```

```
                if candidate.ballot not in stats['ballots']:
                    stats['ballots'].append(candidate.ballot)

        # convert vote_ids into Vote objects
        for vote_id, points in vote_points.iteritems():
            vote = Vote.objects.get(id=vote_id)
            stats['votes'][vote] = points

        return stats

    def disassociate_accounts(self):
        """
        Sets account = NULL for all Vote objects associated with this election.
        Returns number of rows affected.
        """
        from django.db import connection
        cursor = connection.cursor()
        query = """
            UPDATE %(vote)s
            SET account_id = NULL
            WHERE election_id = %(id)i
        """
        cursor.execute(query % {
            'vote': Vote._meta.db_table,
            'id': self.pk,
        })
        return cursor.rowcount

    @staticmethod
    def get_latest_or_404():
        """
        Similar to the get_object_or_404() function, except returns the
        latest Election instead.
        """
        try:
            return Election.objects.latest()
        except Election.DoesNotExist:
            raise Http404("No elections have been entered yet.")

    class Meta:
        ordering = ['vote_start']
        get_latest_by = "vote_start"


class Ballot(models.Model):
    """
    Represents a ballot of a certain type (plurality, etc.) for an election.
    Each ballot has one or more candidates associated with it.
    """
    TYPES = (
        ("Pr", "Preferential"),
        # ("Pl", "Plurality"),
    )
    election = models.ForeignKey(Election, related_name="ballots")
```

```
position_number = models.PositiveSmallIntegerField(default=1,
    help_text="Change this if you want to customize the order in which "+\
    "ballots are shown for an election.")
description = models.CharField(max_length=255, blank=True)
introduction = models.TextField(blank=True,
    help_text="If this field is non-empty, it will be shown below the "+\
    "ballot header on the voting page")
type = models.CharField(max_length=2, blank=False, choices=TYPES,
    default="Pr", help_text="Ballots are preferential by default")
#seats_available = models.PositiveSmallIntegerField()
#is_secret = models.BooleanField(default=False,
 #   help_text="Check this for a secret ballot. This means that only the "+\
  #  "fact that a voter voted will be recorded, not his or her choices.")
write_in_available = models.BooleanField(default=False)

def __unicode__(self):
    return "%s %s: %s" % (self.get_type_display(),
                          unicode(self.election), self.description)

def get_candidate_stats(self):
    """
    Returns list of form [(candidate1, x1), (candidate2, x2), ...]
    where x1, x2, ... are the total number of votes if self.type == "Pl"
    or the sum of the point values if self.type == "Pr"
    """
    stats = []
    if self.type == "Pl":
        stats = [(c, VotePlurality.objects.filter(candidate=c).count())
                 for c in self.candidates.all()]
        # sort according to # of votes, in reverse order
        stats.sort(key=lambda x: x[1], reverse=True)
    elif self.type == "Pr":
        #fall back to SQL since Django DB API doesn't support sum()
        from django.db import connection
        cursor = connection.cursor()
        query = """
            SELECT c.id, IF(v.point IS NULL, 0, SUM(v.point)) AS points
            FROM %(ballot)s b
            JOIN %(candidate)s c ON (c.ballot_id=b.id)
            LEFT JOIN %(vote_preferential)s v ON (v.candidate_id=c.id)
            WHERE b.id = '%(id)i'
            GROUP BY c.id
            ORDER BY points DESC
        """
        cursor.execute(query % {
            'ballot': Ballot._meta.db_table,
            'candidate': Candidate._meta.db_table,
            'vote_preferential': VotePreferential._meta.db_table,
            'id': self.pk,
        })
        stats = [(Candidate.objects.get(id=c[0]), c[1])
                 for c in cursor.fetchall()]
    return stats
```

```
    def candidates_with_biographies(self):
        return self.candidates.exclude(biography="")

    def has_incumbents(self):
        """
        Returns True if ballot has any candidates associated with it that
        are incumbents
        """
        return self.candidates.filter(incumbent=True).count() > 0

    class Meta:
        ordering = ['election', 'position_number', 'type', 'description']


class Candidate(models.Model):
    """
    Model for election candidates (e.g. governing board, nominating committee).
    Each candidate must be associated with a single ballot.
    """
    ballot = models.ForeignKey(Ballot, related_name="candidates")
    first_name = models.CharField(max_length=255, blank=False)
    last_name = models.CharField(max_length=255, blank=False)
    institution = models.CharField(max_length=255, blank=True)
    # incumbent = models.BooleanField(default=False)
    # image_url = models.URLField(max_length=255, blank=True)
    picture = models.ImageField('Profile picture',
                                #upload_to="static/django_elect/img",
                                upload_to='candidate_pics/%d-%m-%Y/',
                                validators=[validate_image_extension],
                                null=True,
                                blank=True)
    write_in = models.BooleanField(default=False)
    biography = models.TextField(blank=True,
        help_text="Enter the candidate's biography here. It will "+\
        "be shown when the user clicks the candidate's name. If you leave "+\
        "this field blank, the candidate's name will not be a link.")

    def __unicode__(self):
        # parenthesis = self.institution or (self.write_in and "write-in")
        return "%s %s" % (self.first_name, self.last_name)

    def get_name(self):
        """Returns full name of candidate."""
        return self.first_name+" "+self.last_name

    class Meta:
        ordering = ['last_name', 'first_name']


class Vote(models.Model):
    """
    Vote associates individual candidate selections with an account and
    an election.
    """
```

```
    account = models.ForeignKey(settings.DJANGO_ELECT_USER_MODEL, null=True)
    election = models.ForeignKey(Election, related_name="votes")

    def __unicode__(self):
        return unicode(self.account) + " - " + unicode(self.election)

    def get_details(self):
        """
        Returns list in form
        [ (ballot1, [vote1, vote2, ...]), (ballot2, [vote3, vote4, ...]), ...]
        """
        details = []
        for ballot in self.election.ballots.all():
            votes = []
            if ballot.type == "Pl":
                votes = (self.pluralities
                             .filter(candidate__ballot=ballot)
                             .order_by('candidate'))
            elif ballot.type == "Pr":
                votes = (self.preferentials
                             .filter(candidate__ballot=ballot)
                             .order_by('candidate'))
            details.append((ballot, votes))
        return details


def _get_choices(ballot_type):
    """
    Returns Q object that matches a ballot of the specified type that's
    currently active, i.e. now() is between the vote_start and vote_end dates
    """
    return Q(ballot__type=ballot_type) &\
           Q(ballot__election__vote_start__lte=datetime.now()) &\
           Q(ballot__election__vote_end__gte=datetime.now())


class VotePreferential(models.Model):
    """
    Vote for a candidate on a preferential ballot (i.e. Ballot.type="Pr")
    """
    vote = models.ForeignKey(Vote, related_name="preferentials", null=True)
    candidate = models.ForeignKey(Candidate,
        limit_choices_to=_get_choices("Pr"))
    point = models.PositiveSmallIntegerField()

    def __unicode__(self):
        return "%s vote, %i points for %s" % (self.vote, self.point,
                                              self.candidate.get_name())


class VotePlurality(models.Model):
    """
    Vote for a candidate on a plurality ballot (i.e. Ballot.type="Pl")
    """
```

```
    vote = models.ForeignKey(Vote, related_name="pluralities", null=True)
    candidate = models.ForeignKey(Candidate,
        limit_choices_to=_get_choices("P1"))

    def __unicode__(self):
        return "%s vote for %s" % (self.vote, self.candidate.get_name())
```

4.6 settings.py

```
from django.conf import settings
from django.db.models.functions import Concat
from django.db.models import Value as V

"""
A string that corresponds to the path to the model that should be used for
the Election.allowed_voters and Vote.account foreign keys. This is mainly for
sites that extend the User model via inheritance, as detailed at
http://scottbarnham.com/blog/2008/08/21/extending-the-django-user-model-with-inheritance/
"""
DJANGO_ELECT_USER_MODEL = getattr(settings,
    'DJANGO_ELECT_USER_MODEL', settings.AUTH_USER_MODEL)


"""
Function to filter a queryset on the user model with a free-form query.
Used by django_elect.autocomplete.AccountAutocomplete.
"""
DJANGO_ELECT_USER_AUTOCOMPLETE_FILTER = getattr(settings,
    'DJANGO_ELECT_USER_AUTOCOMPLETE_FILTER',
    lambda queryset, query: queryset.annotate(
        full_name=Concat('first_name', V(' '), 'last_name')
    ).filter(full_name__icontains=query))


"""
List of tuples to pass to Migration.depedencies for django_elect migrations
"""
DJANGO_ELECT_MIGRATION_DEPENDENCIES = getattr(settings,
    'DJANGO_ELECT_MIGRATION_DEPENDENCIES', [('auth', '0001_initial')])


"""
URL to redirect voters to who are not logged in.
"""
LOGIN_URL = getattr(settings, 'LOGIN_URL', '/account/')
```

4.7 urls.py

```
from django.conf.urls import patterns, url
from django_elect import views, autocomplete

urlpatterns = patterns('',
    url(r'^$', views.HomePage.as_view(), name='home'),
```

```
    url(r'^biographies', views.biographies, name="django_elect_biographies"),
    url(r'^success', views.success, name="django_elect_success"),
    url(r'^statistics/(?P<id>\d+)', views.statistics,
        name="django_elect_stats"),
    url(r'^spreadsheet/(?P<id>\d+)', views.generate_spreadsheet,
        name="django_elect_spreadsheet"),
    url(r'^disassociate/(?P<id>\d+)', views.disassociate_accounts,
        name="django_elect_disassociate"),
    url(r'^vote-plurality-autocomplete/$',
        autocomplete.CandidateAutocomplete.as_view(ballot_type="Pl"),
        name='vote-plurality-autocomplete'),
    url(r'^vote-preferential-autocomplete/$',
        autocomplete.CandidateAutocomplete.as_view(ballot_type="Pr"),
        name='vote-preferential-autocomplete'),
    url(r'^account-autocomplete/$',
        autocomplete.AccountAutocomplete.as_view(),
        name='account-autocomplete'),
    url(r'^vote', views.vote, name="django_elect_vote"),
)
```

4.8 validators.py

```
import os
from django.core.exceptions import ValidationError


def validate_file_extension(value):
    ext = os.path.splitext(value.name)[1]  # [0] returns path+filename
    valid_extensions = ['.xlsx', '.xls']
    if not ext.lower() in valid_extensions:
        raise ValidationError(u'Unsupported file extension.')


def validate_image_extension(value):
    ext = os.path.splitext(value.name)[1]  # [0] returns path+filename
    valid_extensions = ['.JPEG', '.JPG', '.PNG', '.jpeg', '.jpg', '.png']
    if not ext.lower() in valid_extensions:
        raise ValidationError(u'Unsupported image extension.')
```

4.9 views.py

```
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response, get_object_or_404, redirect
from django.template import RequestContext
from django.core.urlresolvers import reverse
from django.contrib import messages
from django.contrib.admin.views.decorators import staff_member_required
from django.contrib.auth.decorators import login_required
from django.views.decorators.cache import never_cache
from django.views import generic
from django_elect.models import Election, Vote
from django_elect.forms import PluralityVoteForm, PreferentialVoteForm
from django_elect import settings
```

```
class HomePage(generic.TemplateView):
    template_name = "home.html"


def biographies(request):
    election = Election.get_latest_or_404()
    ballot_candidates = dict((b, b.candidates_with_biographies())
        for b in election.ballots.all() if b.candidates_with_biographies())
    return render_to_response('django_elect/biographies.html', {
        'election': election,
        'ballot_candidates': ballot_candidates.items(),
    })


@staff_member_required
@never_cache
def statistics(request, id):
    """
    Displays a table for each ballot with statistics for the candidates.
    """
    election = get_object_or_404(Election, pk=id)
    return render_to_response('django_elect/statistics.html', {
        'title': "Election Statistics",
        'election': election,
    })


@staff_member_required
def generate_spreadsheet(request, id):
    """
    Generates an Excel spreadsheet for review by a staff member.
    """
    election = get_object_or_404(Election, pk=id)
    response = render_to_response("django_elect/spreadsheet.html", {
        'full_stats': election.get_full_statistics(),
    })
    filename = "election%s.xls" % (election.pk)
    response['Content-Disposition'] = 'attachment; filename='+filename
    response['Content-Type'] = 'application/vnd.ms-excel; charset=utf-8'
    return response


@staff_member_required
def disassociate_accounts(request, id):
    """
    Disassociates accounts (i.e. sets account_ids to NULL) for all Vote
    objects. 'id' corresponds to the primary key of the Election objects.
    """
    election = get_object_or_404(Election, pk=id)
    success = False
    if request.POST and "confirm" in request.POST:
        election.disassociate_accounts()
        success = True
```

```
    return render_to_response("django_elect/disassociate.html", {
        "title": "Disassociate Accounts for Election %s" % election,
        "election": election,
        "success": success,
    }, context_instance=RequestContext(request))


@login_required
def vote(request):
    if request.user.is_staff:
        messages.error(request, "Staff are not allowed to vote!")
        return redirect('/')

    election = Election.get_latest_or_404()
    if not election.voting_allowed_for_user(request.user):
        messages.error(request, "You have already voted!")
        # they aren't supposed to be on this page
        return redirect('/')

    forms = []
    none_selected = False
    data = request.POST or None
    # fill forms list with Form objects, one for each ballot
    for b in election.ballots.all():
        prefix = "ballot%i" % (b.id)
        if b.type == "Pl":
            form = PluralityVoteForm(b, data=data, prefix=prefix)
        elif b.type == "Pr":
            form = PreferentialVoteForm(b, data=data, prefix=prefix)
        forms.append(form)

    if request.POST and all(x.is_valid() for x in forms):
        #all forms valid, so save unless no candidates were selected
        if any(f.has_candidates() for f in forms):
            vote = election.create_vote(request.user)
            for f in forms:
                f.save(vote)
            messages.success(request, "You have successfully voted. Thank you.")
            return redirect('/')
            #return HttpResponseRedirect(reverse("voting:django_elect_success"))
        else:
            # they must not have selected any candidates, so show an error
            none_selected = True

    return render_to_response('django_elect/vote.html', {
        'current_tab': 'election',
        'account': request.user,
        'election': election,
        'forms': forms,
        'none_selected': none_selected,
    }, context_instance=RequestContext(request))
def success(request):
    return render_to_response('django_elect/success.html')
```

4.10 django_elect templates

4.10.1 base.html

```
{% load staticfiles %}
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>{% block title %}{% endblock %}</title>
        <link rel="stylesheet" type="text/css"
              href="{% static "django_elect/css/main.css" %}"/>
        {% block extrahead %}{% endblock %}
    </head>
    <body>
        <div id="header">
          <a name="top" id="top"></a>
          <h1 id="dept"><a href="index.html">Test</a></h1>
        </div>
        <div id="mainContainer">
          <div id="content">
            {% block content %}{% endblock %}
          </div>
        </div>
    </body>
</html>
```

4.10.2 errors.html

```
{% load staticfiles %}
<div class="notice" id="error{{id}}">
    <a name="error{{id}}"></a>
    <img src="{% static "django_elect/img/error.gif" %}" alt="Error"/>
    Please correct the errors below and try again.
    <ul class="error_list">
    {% for error_list in error_dict %}
        {% for error in error_list.1 %}
        <li>
            {% if error_list.0 %}
            <div class="error_message"> {{error_list.0}}:</div>
            {% endif %}
            {{error}}
        </li>
        {% endfor %}
    {% endfor %}
    </ul>
</div>
```

4.10.3 statistics.html

```
{% extends "admin/change_form.html" %}
{% block breadcrumbs %}
<div class="breadcrumbs">
  <a href="/admin/">Home</a> &rsaquo; Election Statistics
</div>
{% endblock %}
{% block content %}
<div id="content-main">
  <h1>Statistics for {{election}} Election</h1>
  {% for ballot in election.ballots.all %}
  <h2>Ballot {{ballot}}</h2>
  <table cellspacing="0" cellpadding="0" border="1">
    <tr>
      <th>Candidate</th>
      <th>
        Total {% ifequal ballot.type "Pl" %}Votes{% else %}Points{% endifequal %}
      </th>
    </tr>
    {% for candidate,result in ballot.get_candidate_stats %}
      <tr>
        <td>{{candidate}}</td>
        <td>{{result}}</td>
      </tr>
    {% endfor %}
  </table>
  <br/>
  {% endfor %}
</div>
{% endblock %}
```

4.10.4 success.html

```
{% extends "django_elect/base.html" %}
{% block title%}Success{% endblock %}
{% block content %}
<div class="section">
    <div class="heading">
      <h2>Success</h2>
    </div>
    <div class="content">
      You have successfully voted. Thank you.
    </div>
</div>
{% endblock %}
```

4.10.5 vote.html

```
{% extends "base.html" %}
{% load staticfiles %}
{% load show_errors %}
{% block title %}{{ block.super }}{{election}} Election - Vote{% endblock %}
```

```
{% block navbar-left %}
  {% include "_navbar.html" with active_link="vote" %}
{% endblock %}
{% block container %}
<div class="content-section">
  <div class="container">
    <div class="row"><br><br><br>
      <div class="col-sm-3"></div>
        <div class="col-sm-6 text-center">
          <form method="post" action="">{% csrf_token %}
            <h2>{{election}} Election</h2>
            <p>
              {{election.introduction|safe}}
            </p><br>
            {% if none_selected %}
            <div class="notice" id="error0">
                <img src="{% static "django_elect/img/error.gif" %}" alt="Error" />
                Please select at least one candidate.
            </div>
            {% endif %}
          {% for form in forms %}
            <hr class="style3">
            <div class="section" style="background-color:lightblue;">
              <div class="heading">
                <h3><span
class="ballot-heading"><b>{{form.ballot.description}}</b></span></h3>
                {% if form.ballot.introduction %}
                <h6><span>{{form.ballot.introduction|safe}}</span></h6>
                {% endif %}
              </div>
              <div class="content">
                {% if form.errors %}
                  {% show_errors form %}
                  <script type="text/javascript">
                  window.location.hash = "error0";
                  </script>
                {% endif %}
                {{form}}
              </div>
              Number each candidate from 1<br>
              <b>(1 = Most preferred)</b><br><br>
            </div>
          {% endfor %}
            <hr class="style3"><br>
            <p><input type="submit" class="btn btn-lg btn-success btn-block" name="vote"
onclick="return confirm('Are you sure?')" value="Submit Vote"/></p>
          </form>
        </div>
      <div class="col-sm-3"></div>
    </div>
  </div>
</div>
{% endblock %}
```

4.10.6 vote_details.html

```
<table cellspacing="0" cellpadding="3">
  <tr>
    <th><h4>Ballot</h4></th>
    <th><h4>Candidates Selected</h4></th>
  </tr>
{% for ballot,candidateVotes in vote.get_details %}
  <tr>
    <td><p>{{ballot.description}}</p></td>
    <td>
      <p>
      {% if not candidateVotes %}
        None
      {% else %}
        {% for cv in candidateVotes %}
          {{cv.candidate.get_name}}{% if cv.point %}( {{cv.point}} points ){% endif %}{% if
not forloop.last %},{% endif %}
        {% endfor %}
      {% endif %}
      </p>
    </td>
  </tr>
{% endfor %}
</table>
```

5. evoteproject

This section contains all the project files.

5.1 logger.py

```
import logging

class NewStyleLogMessage(object):
    def __init__(self, message, *args, **kwargs):
        self.message = message
        self.args = args
        self.kwargs = kwargs

    def __str__(self):
        args = (i() if callable(i) else i for i in self.args)
        kwargs = dict((k, v() if callable(v) else v)
                      for k, v in self.kwargs.items())

        return self.message.format(*args, **kwargs)

N = NewStyleLogMessage

class StyleAdapter(logging.LoggerAdapter):
    def __init__(self, logger, extra=None):
```

```
        super(StyleAdapter, self).__init__(logger, extra or {})

    def log(self, level, msg, *args, **kwargs):
        if self.isEnabledFor(level):
            msg, log_kwargs = self.process(msg, kwargs)
            self.logger._log(level, N(msg, *args, **kwargs), (),
                             **log_kwargs)


logger = StyleAdapter(logging.getLogger("project"))
#   Emits "Lazily formatted log entry: 123 foo" in log
# logger.debug('Lazily formatted entry: {0} {keyword}', 123, keyword='foo')
```

5.2 urls.py

```
from django.conf.urls import include, url
from django.contrib import admin
from django.conf import settings
from django.conf.urls.static import static
import accounts.urls
from . import views

urlpatterns = [
    url(r'^$', views.HomePage.as_view(), name='home'),
    url(r'^about/$', views.AboutPage.as_view(), name='about'),
    url(r'^admin/', include(admin.site.urls)),
    url(r'^', include(accounts.urls, namespace='accounts')),
    url(r'^election/', include('django_elect.urls', namespace='voting')),
]

# User-uploaded files like profile pics need to be served in development
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

# Include django debug toolbar if DEBUG is on
if settings.DEBUG:
    import debug_toolbar
    urlpatterns += [
        url(r'^__debug__/', include(debug_toolbar.urls)),
    ]
```

5.3 views.py

```
from django.views import generic

class HomePage(generic.TemplateView):
    template_name = "home.html"


class AboutPage(generic.TemplateView):
    template_name = "about.html"
```

5.4 evoteproject settings

5.4.1 base.py

```
from django.core.urlresolvers import reverse_lazy
from os.path import dirname, join, exists

# Build paths inside the project like this: join(BASE_DIR, "directory")
BASE_DIR = dirname(dirname(dirname(__file__)))
STATICFILES_DIRS = [join(BASE_DIR, 'static')]
MEDIA_ROOT = join(BASE_DIR, 'media')
MEDIA_URL = "/media/"

# Use Django templates using the new Django 1.8 TEMPLATES settings
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            join(BASE_DIR, 'templates'),
            # insert more TEMPLATE_DIRS here
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                # Insert your TEMPLATE_CONTEXT_PROCESSORS here or use this
                # list if you haven't customized them:
                'django.contrib.auth.context_processors.auth',
                'django.template.context_processors.debug',
                'django.template.context_processors.i18n',
                'django.template.context_processors.media',
                'django.template.context_processors.static',
                'django.template.context_processors.tz',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

# Use 12factor inspired environment variables or from a file
import environ
env = environ.Env()

# Ideally move env file should be outside the git repo
# i.e. BASE_DIR.parent.parent
env_file = join(dirname(__file__), 'local.env')
if exists(env_file):
    environ.Env.read_env(str(env_file))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/dev/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
```

```
# Raises ImproperlyConfigured exception if SECRET_KEY not in os.environ
SECRET_KEY = env('SECRET_KEY')

SECURE_SSL_REDIRECT = False  # Must be set to True for production

ALLOWED_HOSTS = []


INSTALLED_APPS = (
    'django_elect',
    'dal',
    'dal_select2',

    'django.contrib.auth',
    'django_admin_bootstrapped',
    'django.contrib.admin',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'authtools',
    'crispy_forms',
    'easy_thumbnails',
    'accounts',
)

MIDDLEWARE_CLASSES = (
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)

ROOT_URLCONF = 'evoteproject.urls'

WSGI_APPLICATION = 'evoteproject.wsgi.application'

# Database
# https://docs.djangoproject.com/en/dev/ref/settings/#databases

DATABASES = {
    # Raises ImproperlyConfigured exception if DATABASE_URL not in
    # os.environ
    'default': env.db(),
}

EMAIL_BACKEND = "sgbackend.SendGridBackend"
SENDGRID_API_KEY = env('SENDGRID_API_KEY')
```

```
# Email settings
#EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
#EMAIL_HOST = env('EMAIL_HOST')
#EMAIL_PORT = env('EMAIL_PORT')
#EMAIL_HOST_USER = env('EMAIL_HOST_USER')
#EMAIL_HOST_PASSWORD = env('EMAIL_HOST_PASSWORD')
#EMAIL_USE_TLS = env('EMAIL_USE_TLS')

# Internationalization
# https://docs.djangoproject.com/en/dev/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = False

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/dev/howto/static-files/

STATIC_URL = '/static/'

#ALLOWED_HOSTS = []

# Crispy Form Theme - Bootstrap 3
CRISPY_TEMPLATE_PACK = 'bootstrap3'

# For Bootstrap 3, change error alert to 'danger'
from django.contrib import messages
MESSAGE_TAGS = {
    messages.ERROR: 'danger'
}

# Authentication Settings
AUTH_USER_MODEL = 'authtools.User'
LOGIN_REDIRECT_URL = 'home'
LOGIN_URL = reverse_lazy("accounts:login")
LOGOUT_REDIRECT_URL = 'home'

THUMBNAIL_EXTENSION = 'png'     # Or any extn for your thumbnails

DEBUG_TOOLBAR_CONFIG = {
    'SHOW_TOOLBAR_CALLBACK': lambda r: False,  # disables it
    # '...
}

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
```

```
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
        'OPTIONS': {
            'min_length': 8,
        }
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

5.4.2 development.py

```
from .base import *
import sys
import logging.config

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
TEMPLATES[0]['OPTIONS'].update({'debug': True})

# Turn off debug while imported by Celery with a workaround
# See http://stackoverflow.com/a/4806384
if "celery" in sys.argv[0]:
    DEBUG = False

# Django Debug Toolbar
INSTALLED_APPS += (
    'debug_toolbar',)

# Additional middleware introduced by debug toolbar
MIDDLEWARE_CLASSES += (
    'debug_toolbar.middleware.DebugToolbarMiddleware',)

# Show emails to console in DEBUG mode
# EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'

# Show thumbnail generation errors
THUMBNAIL_DEBUG = True

# Allow internal IPs for debugging
INTERNAL_IPS = [
    '127.0.0.1',
    '0.0.0.1',
]

# Log everything to the logs directory at the top
LOGFILE_ROOT = join(dirname(BASE_DIR), 'logs')

# Reset logging
# (see
```

http://www.caktusgroup.com/blog/2015/01/27/Django-Logging-Configuration-logging_config-default-settings-logger/)

```
LOGGING_CONFIG = None
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': "[%(asctime)s] %(levelname)s [%(pathname)s:%(lineno)s] %(message)s",
            'datefmt': "%d/%b/%Y %H:%M:%S"
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'django_log_file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': join(LOGFILE_ROOT, 'django.log'),
            'formatter': 'verbose'
        },
        'proj_log_file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': join(LOGFILE_ROOT, 'project.log'),
            'formatter': 'verbose'
        },
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'simple'
        }
    },
    'loggers': {
        'django': {
            'handlers': ['django_log_file'],
            'propagate': True,
            'level': 'DEBUG',
        },
        'project': {
            'handlers': ['proj_log_file'],
            'level': 'DEBUG',
        },
    }
}

logging.config.dictConfig(LOGGING)
```

5.4.3 local.env

```
DEBUG=True
EMAIL_HOST=smtp.sendgrid.net
```

```
EMAIL_PORT=587
EMAIL_HOST_USER=marcuskelly
EMAIL_HOST_PASSWORD=R0ck3t0namaz3
EMAIL_USE_TLS=True
SENDGRID_API_KEY=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
DATABASE_URL=mysql://root:r0ck3t@127.0.0.1:/evote
# DATABASE_URL=sqlite:///db.sqlite3
# Command to create a new secret key:
# $ python -c 'import random; import string;
print("".join([random.SystemRandom().choice(string.digits + string.ascii_letters +
string.punctuation) for i in range(100)]))'
SECRET_KEY=+j*8(*pu8k_^)ie5qz8=$ose#y-16+!2=4^#=u%rzjoup)kjcr
```

5.4.4 production.py

```
from .base import *
import logging.config

# For security and performance reasons, DEBUG is turned off
DEBUG = False
TEMPLATE_DEBUG = False

# Must mention ALLOWED_HOSTS in production!
ALLOWED_HOSTS = [u'itcsuvoting.pythonanywhere.com']

# Cache the templates in memory for speed-up
loaders = [
    ('django.template.loaders.cached.Loader', [
        'django.template.loaders.filesystem.Loader',
        'django.template.loaders.app_directories.Loader',
    ]),
]

TEMPLATES[0]['OPTIONS'].update({"loaders": loaders})
TEMPLATES[0].update({"APP_DIRS": False})

# Define STATIC_ROOT for the collectstatic command
STATIC_ROOT = join(BASE_DIR, '/home/itcsuvoting/evote/static', 'site', 'static')

# STATIC_ROOT = 'static'

# Log everything to the logs directory at the top
LOGFILE_ROOT = join(dirname(BASE_DIR), 'logs')

# Reset logging
LOGGING_CONFIG = None
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        'verbose': {
            'format': "[%(asctime)s] %(levelname)s [%(pathname)s:%(lineno)s] %(message)s",
            'datefmt': "%d/%b/%Y %H:%M:%S"
        },
```

```
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'proj_log_file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': join(LOGFILE_ROOT, 'project.log'),
            'formatter': 'verbose'
        },
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'simple'
        }
    },
    'loggers': {
        'project': {
            'handlers': ['proj_log_file'],
            'level': 'DEBUG',
        },
    }
}

logging.config.dictConfig(LOGGING)
```

6. WSGI configuration file

```
import os
import sys

path = '/home/itcsuvoting/evote/evoteproject/src'
if path not in sys.path:
    sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'evoteproject.settings'
SECRET_KEY=os.environ["SECRET_KEY"]

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```