```
from django.shortcuts import render, redirect
from datetime import datetime, timedelta
import requests
import math
from Crypto.Cipher import AES
from django.contrib.auth import authenticate, logout, login
from django.contrib.auth.decorators import login_required


from .models import Member, Job, Encrypted_Client, Encrypted_Job, HomeAddress
from .forms import MemberForm, JobForm, EditMemberForm, EditJobForm,
LoginForm, ChangePassword, ChangeHome



apiKey = 'AIzaSyBkI0mUe0rTnpQd9V4pTq-V20GkU5_d5RY'



def pad(text):
    text = text.replace('q', 'qa')
    text = text.replace('x', 'qb')

    padlength = 16 - (len(text) % 16)

    for m in range(padlength):
        text += 'x'

    return text



def unpad(text):
    text = text.replace('x', '')
    text = text.replace('qb', 'x')
    text = text.replace('qa', 'q')

    return text



def encrypt(message, key, iv):

    message = pad(message)
    key = pad(key)
    iv = pad(iv)

    obj = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = obj.encrypt(message)
    return ciphertext



def decrypt(ciphertext, key, iv):

    key = pad(key)
    iv = pad(iv)

    obj = AES.new(key, AES.MODE_CBC, iv)
    message = obj.decrypt(ciphertext).decode()

    message = unpad(message)

    return message



def get_clients():

    encrypted_clients = Encrypted_Client.objects.order_by('-date_added')
    decrypted_clients = []

    for e in encrypted_clients:
        decrypted_client = Member(fname = decrypt(e.fname, 'password', str(8)
+ 'fname'),
            lname = decrypt(e.lname, password, str(8) + 'lname'),
            email = decrypt(e.email, password, str(8) + 'email'),
            address = decrypt(e.address, password, str(8) + 'address'),
            homephone = decrypt(e.homephone, password, str(8) + 'homephone'),
            acreage = decrypt(e.acreage, password, str(8) + 'acreage'),
            client_id = e.id
            )
        decrypted_clients.append(decrypted_client)

    return decrypted_clients



def get_client(cid):

    e = Encrypted_Client.objects.filter(id=cid)[0]

    decrypted_client = Member(fname = decrypt(e.fname, 'password', str(8) +
'fname'),
        lname = decrypt(e.lname, password, str(8) + 'lname'),
        email = decrypt(e.email, password, str(8) + 'email'),
        address = decrypt(e.address, password, str(8) + 'address'),
        homephone = decrypt(e.homephone, password, str(8) + 'homephone'),
        acreage = decrypt(e.acreage, password, str(8) + 'acreage')
        )

    return decrypted_client



def get_jobs():
    encrypted_jobs = Encrypted_Job.objects.order_by('day', 'time')
    decrypted_jobs = []

    clients = get_clients()

    for j in encrypted_jobs:
        job_client = ''
        for c in clients:
            if j.client.id == c.client_id:
                job_client = c
        decrypted_job = Job(
            client = job_client,
            day = j.day,
            time = j.time,
            jtype = j.jtype,
            start = j.start,
            hours = j.hours,
            minutes = j.minutes,
            description = j.description,
            job_id = j.id
            )
        decrypted_jobs.append(decrypted_job)

    return decrypted_jobs



def get_clientnamelist():
    clients = get_clients()
    clientnamelist = []

    for c in clients:
        clientnamelist.append((c.client_id, c.lname))

    return clientnamelist



def get_drive_data(origin, destination):

    url =
('https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial&ori
gins={}&destinations={}&key={}'
```

```
           .format(origin.replace(' ','+'),
                   destination.replace(' ','+'),
                   apiKey
                  )
          )

    valid = True

    try:
        response = requests.get(url)
        resp_json_payload = response.json()
        drive_distance = 
resp_json_payload['rows'][0]['elements'][0]['distance']['value']
        drive_time = 
resp_json_payload['rows'][0]['elements'][0]['duration']['value']

    except:
        valid = False
        drive_distance = 0
        drive_time = 0
    return {'valid': valid, 'distance': drive_distance, 'time': drive_time}



def valid_user(req):
    username = 'no username'
    if req.session.has_key('username'):
        username = req.session['username']
    return username


def get_travel_time(address_A, address_B):
    return timedelta(seconds=get_drive_data(address_A, address_B)['time'])



def do_jobs_overlap(jobs, new_job, editing):

    if editing:

        cleaned_jobs = []
        for j in jobs:
            if not j.job_id == new_job['id']:
                cleaned_jobs.append(j)

        jobs = cleaned_jobs

    nj_begin = datetime.combine(new_job['day'], new_job['time'])
    nj_duration = timedelta(hours = new_job['hours'], minutes = 
new_job['minutes'])
```

```
    nj = {
        'begin': nj_begin,
        'end': nj_begin + nj_duration
        }

    job_boundaries = []

    for j in jobs:

        begin = datetime.combine(j.day, j.time)
        td = timedelta(hours = j.hours, minutes = j.minutes)
        end = begin + td

        job_boundaries.append({'job': j, 'begin': begin, 'end': end,
'status': 'n'})


    for j in job_boundaries:

        if nj['end'] == j['begin']:
            return 'travel'
        if nj['end'] > j['begin']:
            if nj['begin'] < j['end']:
                return 'overlap'
            if nj['begin'] == j['end']:
                return 'travel'
            j['status'] = 'before'
        else:
            j['status'] = 'after'

    prev_job = ''
    next_job = ''

    for j in range(len(job_boundaries[0:-1])):
        if job_boundaries[j]['status'] == 'before' and job_boundaries[j +
1]['status'] == 'after':
            prev_job = job_boundaries[j]
            next_job = job_boundaries[j + 1]


    travelAC = ''

    if prev_job['job'].client.address == new_job['start']:
        travelAC = get_travel_time(prev_job['job'].client.address,
new_job['address'])
    else:
        travelAB = get_travel_time(prev_job['job'].client.address,
new_job['start'])
        travelBC = get_travel_time(new_job['start'], new_job['address'])
        travelAC = travelAB + travelBC
    if prev_job['end'] + travelAC > nj['begin']:
        return 'travel'


    travelCE = ''

    if prev_job['job'].client.address == next_job['job'].start:
        travelCE = get_travel_time(new_job['address'],
next_job['job'].client.address)
    else:
        travelCD = get_travel_time(new_job['address'], next_job['job'].start)
        travelDE = get_travel_time(next_job['job'].start,
next_job['job'].client.address)
        travelCE = travelCD + travelDE

    if nj['end'] + travelCE > next_job['begin']:
        return 'travel'


    return 'good'


def split_address(full_address):

    split_address = full_address.split(',')

    if len(split_address) == 3:

        street = split_address[0].strip()
        city = split_address[1].strip()
        state = split_address[2].strip().upper()

    elif len(split_address) == 2:
        street = split_address[0].strip()
        city = split_address[1].strip()
        state = 'FL'

    else:
        street = split_address[0].strip()
        city = 'none'
        state = 'FL'

    return [street, city, state]



@login_required
def clients(request):
    clients = get_clients()

    memberbucket = []


    for c in clients:

        address_list = split_address(c.address)

        newform = EditMemberForm(initial={
            'client_id': c.client_id,
            'fname': c.fname,
            'lname': c.lname,
            'email': c.email,
            'street': address_list[0],
            'city': address_list[1],
            'state': address_list[2],
            'homephone': c.homephone,
            'acreage': c.acreage,
            })

        error = {'field': 'none', 'message': 'none'}

        memberbucket.append({'client': c, 'editform': newform, 'error':
error})

    form = MemberForm()

    context = {'form': form, 'memberbucket': memberbucket, 'error': 'none'}

    return render(request, 'lawncaresite/clients.html', context)



def add_client(request):
    form = MemberForm(request.POST)

    if form.is_valid():

        full_address = request.POST['street'] + ', ' + request.POST['city'] +
', ' + request.POST['state']
        full_address2 = request.POST['street'] + ', ' + request.POST['city']

        home = HomeAddress.objects.all()[0].address;
        if get_drive_data(home, full_address)['valid'] and
get_drive_data(home, full_address2)['valid']:

            new_client =
Encrypted_Client(fname=encrypt(request.POST['fname'], 'password', str(8) +
'fname'),
                lname=encrypt(request.POST['lname'], 'password', str(8) +
'lname'),
                email=encrypt(request.POST['email'], 'password', str(8) +
'email'),
                address=encrypt(full_address, 'password', str(8) +
'address'),
                homephone=encrypt(request.POST['homephone'], 'password',
str(8) + 'homephone')
                )
            new_client.save()
            return redirect('clients')

        else:
            err = 'address'
    else:
        err = 'invalid'

    clients = get_clients()

    memberbucket = []

    for c in clients:

        address_list = split_address(c.address)

        newform = EditMemberForm(initial={
            'client_id': c.client_id,
            'fname': c.fname,
            'lname': c.lname,
            'email': c.email,
            'street': address_list[0],
            'city': address_list[1],
            'state': address_list[2],
            'homephone': c.homephone,
        })

        error = {'field': 'none', 'message': 'none'}

        memberbucket.append({'client': c, 'editform': newform, 'error':
error})

    returnform = MemberForm(initial={
        'fname': request.POST['fname'],
        'lname': request.POST['lname'],
        'email': request.POST['email'],
        'street': request.POST['street'],
        'city': request.POST['city'],
        'state': request.POST['state'],
        'homephone': request.POST['homephone'],
    })
    error = {'field': 'top', 'message': 'Unknown Error Adding Client'}
    if err == 'invalid':
        error = {'field': 'top', 'message': 'Invalid Field Entry Within 
Client Edit Form'}
    if err == 'address':
        error = {'field': 'top', 'message': 'Address Not Found'}

    context = {'form': returnform, 'memberbucket': memberbucket, 'error': 
error}

    return render(request, 'lawncaresite/clients.html', context)



def edit_client(request):
    form = EditMemberForm(request.POST)

    if form.is_valid():

        full_address = request.POST['street'] + ', ' + request.POST['city'] + 
', ' + request.POST['state']
        full_address2 = request.POST['street'] + ', ' + request.POST['city']

        home = HomeAddress.objects.all()[0].address;
        if get_drive_data(home, full_address)['valid'] and 
get_drive_data(home, full_address2)['valid']:


            this_client = 
Encrypted_Client.objects.filter(id=request.POST['client_id'])[0];
            this_client.fname = encrypt(request.POST['fname'], 'password', 
str(8) + 'fname')
            this_client.lname = encrypt(request.POST['lname'], 'password', 
str(8) + 'lname')
            this_client.email = encrypt(request.POST['email'], 'password', 
str(8) + 'email')
            this_client.address = encrypt(full_address, 'password', str(8) + 
'address')
            this_client.homephone = encrypt(request.POST['homephone'], 
'password', str(8) + 'homephone')
            this_client.save()
            return redirect('clients')

        else:
            err = 'address'
    else:
        err = 'invalid'
    clients = get_clients()

    memberbucket = []



    for c in clients:

        error = {'field': 'none', 'message': 'none'}

        if str(c.client_id) == request.POST['client_id']:

            newform = EditMemberForm(initial={
                'client_id': c.client_id,
                'fname': request.POST['fname'],
                'lname': request.POST['lname'],
                'email': request.POST['email'],
                'street': request.POST['street'],
                'city': request.POST['city'],
                'state': request.POST['state'],
                'homephone': c.homephone,
            })

            if err == 'address':
                error = {'field': 'top', 'message': 'Address Not Found'}
            if err == 'invalid':
                error = {'field': 'top', 'message': 'Invalid Field Entry
Within Client Edit Form'}

        else:

            address_list = split_address(c.address)

            newform = EditMemberForm(initial={
                'client_id': c.client_id,
                'fname': c.fname,
                'lname': c.lname,
                'email': c.email,
                'street': address_list[0],
                'city': address_list[1],
                'state': address_list[2],
                'homephone': c.homephone,
                'acreage': c.acreage,
            })

        memberbucket.append({'client': c, 'editform': newform, 'error':
error})

    returnform = MemberForm()
    error = {'field': 'none', 'message': 'none'}

    context = {'form': returnform, 'memberbucket': memberbucket, 'error':
error}

    return render(request, 'lawncaresite/clients.html', context)



def delete_client(request):
    Encrypted_Client.objects.filter(id=request.POST['client_id']).delete()
    return redirect('clients')



@login_required
def jobs(request):

    jobs = get_jobs()
    clients = get_clients()
    clientnamelist = get_clientnamelist()

    home = HomeAddress.objects.all()[0].address;

    jobbucket = []

    counter = 10000

    for j in jobs:

        counter += 1

        address_list = split_address(j.start)

        newform = EditJobForm(client_choices=clientnamelist, initial={
            'job_id': j.job_id,
            'edit_client': j.client.client_id,
            'edit_day': j.day,
            'edit_time': j.time,
            'edit_type': j.jtype,
            'edit_street': address_list[0],
            'edit_city': address_list[1],
            'edit_state': address_list[2],
            'edit_hours': j.hours,
            'edit_minutes': j.minutes,
            'edit_description': j.description
            })

        timestamp = datetime.combine(j.day, j.time)
        error = {'field': 'none', 'message': 'none'}
        jobbucket.append({'job': j, 'editform': newform, 'error': error,
'timestamp': datetime.timestamp(timestamp), 'counter': counter})

    form = JobForm(client_choices=clientnamelist)

    error = {'field': 'none', 'message': 'none'}

    context = {'jobbucket': jobbucket, 'form': form, 'clients': clients,
'error': error, 'home': home}
    return render(request, 'lawncaresite/jobs.html', context)



def add_job(request):

    clientnamelist = get_clientnamelist()
    form = JobForm(request.POST, client_choices=clientnamelist)
    home = HomeAddress.objects.all()[0].address;
    err = 'none'

    if form.is_valid():

        if form.cleaned_data['hours'] != 0 or form.cleaned_data['minutes'] !=
0:

            job_datetime = datetime.combine(form.cleaned_data['day'],
form.cleaned_data['time'])

            if job_datetime > datetime.now():

                full_start = form.cleaned_data['street'] + ', ' +
form.cleaned_data['city'] + ', ' + form.cleaned_data['state']
                full_start2 = form.cleaned_data['street'] + ', ' +
form.cleaned_data['city']

                if get_drive_data(home, full_start)['valid'] and
get_drive_data(home, full_start2)['valid']:


                    new_job_dict = {
                        'id': 1,
                        'day': form.cleaned_data['day'],
                        'time': form.cleaned_data['time'],
                        'hours': form.cleaned_data['hours'],
                        'minutes': form.cleaned_data['minutes'],
                        'address':
get_client(request.POST['client']).address,
                        'start': full_start
                        }
                    overlap = do_jobs_overlap(get_jobs(), new_job_dict,
False)

                    if overlap == 'good':


                        new_job = Encrypted_Job(

client=Encrypted_Client.objects.filter(id=request.POST['client'])[0],
                            day=request.POST['day'],
                            time=request.POST['time'],
                            jtype=request.POST['type'],
                            start=full_start,
                            hours=request.POST['hours'],
                            minutes=request.POST['minutes'],
                            description=request.POST['description'])
                        new_job.save()
                        return redirect('jobs')


                    else:
                        err = overlap
                else:
                    err = 'address'
            else:
                err = 'past'
        else:
            err = 'zero'
    else:
        err = 'invalid'

    home = HomeAddress.objects.all()[0].address;

    jobs = get_jobs()

    jobbucket = []

    for j in jobs:

        address_list = split_address(j.start)

        newform = EditJobForm(client_choices=clientnamelist, initial={
            'job_id': j.job_id,
            'edit_client': j.client.client_id,
            'edit_day': j.day,
            'edit_time': j.time,
            'edit_type': j.jtype,
            'edit_street': address_list[0],
            'edit_city': address_list[1],
            'edit_state': address_list[2],
            'edit_hours': j.hours,
            'edit_minutes': j.minutes,
            'edit_description': j.description
        })

        timestamp = datetime.combine(j.day, j.time)

        error = {'field': 'none', 'message': 'none'}

        jobbucket.append({'job': j, 'editform': newform, 'error': error,
'timestamp': datetime.timestamp(timestamp)})

    returnform = JobForm(client_choices=clientnamelist, initial={
        'client': request.POST['client'],
        'day': request.POST['day'],
        'time': request.POST['time'],
        'jtype': request.POST['type'],
        'street': request.POST['street'],
        'city': request.POST['city'],
        'state': request.POST['state'],
        'hours': request.POST['hours'],
        'minutes': request.POST['minutes'],
        'description': request.POST['description']
        })

    error = {'field': 'top', 'message': 'Unknown Problem Adding Job'}
    if err == 'invalid':
        error = {'field': 'top', 'message': 'Error: Invalid Form Entry'}
    elif err == 'zero':
        error = {'field': 'minutes', 'message': 'Error: Job duration cannot
be less than 1 minute'}
    elif err == 'past':
        error = {'field': 'day', 'message': 'Error: Jobs cannot be scheduled
in the past'}
    elif err == 'overlap':
        error = {'field': 'top', 'message': 'Error: Job overlaps with an
existing job, consider changing the time or day.'}
    elif err == 'travel':
        error = {'field': 'top', 'message': 'Error: Not enough travel time
between jobs, consider changing the time or day.'}
    if err == 'address':
        error = {'field': 'top', 'message': 'Address Not Found'}

    context = {'form': returnform, 'jobbucket': jobbucket, 'error': error,
'home': home}
    return render(request, 'lawncaresite/jobs.html', context)



def edit_job(request):
    clientnamelist = get_clientnamelist()
    form = EditJobForm(request.POST, client_choices=clientnamelist)
    home = HomeAddress.objects.all()[0].address;
    err = 'none'

    if form.is_valid():

        if form.cleaned_data['edit_hours'] != 0 or
form.cleaned_data['edit_minutes'] != 0:

            job_datetime = datetime.combine(form.cleaned_data['edit_day'],
form.cleaned_data['edit_time'])

            if job_datetime > datetime.now():

                full_start = form.cleaned_data['edit_street'] + ', ' +
form.cleaned_data['edit_city'] + ', ' + form.cleaned_data['edit_state']
                full_start2 = form.cleaned_data['edit_street'] + ', ' +
form.cleaned_data['edit_city']

                if get_drive_data(home, full_start)['valid'] and
get_drive_data(home, full_start2)['valid']:

                    new_job_dict = {
                        'id': form.cleaned_data['job_id'],
                        'day': form.cleaned_data['edit_day'],
                        'time': form.cleaned_data['edit_time'],
                        'hours': form.cleaned_data['edit_hours'],
                        'minutes': form.cleaned_data['edit_minutes'],
                        'address':
get_client(request.POST['edit_client']).address,
                        'start': full_start
                        }
                    overlap = do_jobs_overlap(get_jobs(), new_job_dict, True)

                    if overlap == 'good':

                        this_job =
Encrypted_Job.objects.filter(id=request.POST['job_id'])[0]
                        this_job.client =
Encrypted_Client.objects.filter(id=request.POST['edit_client'])[0]
                        this_job.day = request.POST['edit_day']
                        this_job.time = request.POST['edit_time']
                        this_job.jtype = request.POST['edit_type']
                        this_job.hours = request.POST['edit_hours']
                        this_job.minutes = request.POST['edit_minutes']
                        this_job.start = full_start
                        this_job.description =
request.POST['edit_description']
```

```
                        this_job.save()
                        return redirect('jobs')

                    else:
                        err = overlap
                else:
                    err = 'address'
            else:
                err = 'past'
        else:
            err = 'zero'
    else:
        err = 'invalid'

    home = HomeAddress.objects.all()[0].address;

    jobs = get_jobs()

    jobbucket = []

    for j in jobs:


        error = {'field': 'none', 'message': 'none'}

        if str(j.job_id) == request.POST['job_id']:

            newform = EditJobForm(client_choices=clientnamelist, initial={
                'job_id': request.POST['job_id'],
                'edit_client': request.POST['edit_client'],
                'edit_day': request.POST['edit_day'],
                'edit_time': request.POST['edit_time'],
                'edit_type': request.POST['edit_type'],
                'edit_street': request.POST['edit_street'],
                'edit_city': request.POST['edit_city'],
                'edit_state': request.POST['edit_state'],
                'edit_hours': request.POST['edit_hours'],
                'edit_minutes': request.POST['edit_minutes'],
                'edit_description': request.POST['edit_description']
            })


            error = {'field': 'top', 'message': 'Error: Unknown Problem
Editing Job'}

            if err == 'invalid':
                error = {'field': 'top', 'message': 'Error: Invalid Form
Entry'}
            elif err == 'zero':
                error = {'field': 'minutes', 'message': 'Error: Job duration
cannot be less than 1 minute'}
            elif err == 'past':
                error = {'field': 'day', 'message': 'Error: Jobs cannot be
scheduled in the past'}
            elif err == 'overlap':
                error = {'field': 'top', 'message': 'Error: Job overlaps with
an existing job, consider changing the time or day.'}
            elif err == 'travel':
                error = {'field': 'top', 'message': 'Error: Not enough travel
time between jobs, consider changing the time or day.'}
            elif err == 'address':
                error = {'field': 'top', 'message': 'Error: Address Not
Found'}


        else:

            address_list = split_address(j.start)

            newform = EditJobForm(client_choices=clientnamelist, initial={
                'job_id': j.job_id,
                'edit_client': j.client.client_id,
                'edit_day': j.day,
                'edit_time': j.time,
                'edit_type': j.jtype,
                'edit_street': address_list[0],
                'edit_city': address_list[1],
                'edit_state': address_list[2],
                'edit_hours': j.hours,
                'edit_minutes': j.minutes,
                'edit_description': j.description
            })

        timestamp = datetime.combine(j.day, j.time)

        jobbucket.append({'job': j, 'editform': newform, 'error': error,
'timestamp': datetime.timestamp(timestamp)})


    returnform = JobForm(client_choices=clientnamelist)

    error = {'field': 'none', 'message': 'none'}

    context = {'form': returnform, 'jobbucket': jobbucket, 'error': error,
'home': home}
    return render(request, 'lawncaresite/jobs.html', context)
```

```
def delete_job(request):
    Encrypted_Job.objects.filter(id=request.POST['job_id']).delete()

    return redirect('jobs')


@login_required
def calendar(request, tyear=0, tmonth=0):

    jobs = get_jobs()

    weekdays = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun']
    monthnames = ['null', 'January', 'February', 'March', 'April', 'May', 
'June', 'July',
        'August', 'September', 'October', 'November', 'December']
    last_day = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

    currentmonth = False

    today = datetime.now()


    if tyear == 0 or tyear == today.year:
        tyear = today.year
    if tmonth == 0:
        tmonth = today.month

    if tyear % 4 == 0:
        last_day[2] = 29

    if tyear == today.year and tmonth == today.month:
        currentmonth = True

    tday = today.day

    day_of_month = 0
    day_list = []
    start_numbering = False

    first_weekday = weekdays[datetime(tyear, tmonth, 1).weekday()]


    while not day_of_month >= last_day[tmonth]:

        for wd in weekdays:
            if not start_numbering:
                if wd == first_weekday:
                    day_of_month = 1
                    day_list.append({'dayname': wd, 'dayno': 1, 'jobs': []})
                    start_numbering = True
                else:
                    day_list.append({'dayname': wd, 'dayno': 0, 'jobs': []})
            else:
                if day_of_month < last_day[tmonth]:
                    day_of_month += 1
                    day_list.append({'dayname': wd, 'dayno': day_of_month,
'jobs': []})
                else:
                    day_list.append({'dayname': wd, 'dayno': 0, 'jobs': []})

    for j in jobs:
        if j.day.month == tmonth and j.day.year == tyear:
            for d in day_list:
                if j.day.day == d['dayno']:
                    d['jobs'].append(j)

    calnav = {
        'prev': {'year': '/' + str(tyear), 'month': '/' + str(tmonth - 1)},
        'next': {'year': '/' + str(tyear), 'month': '/' + str(tmonth + 1)}
    }

    if tmonth == 1:
        calnav['prev']['year'] = '/' + str(tyear - 1)
        calnav['prev']['month'] = '/12'
    if tmonth == 12:
        calnav['next']['year'] = '/' + str(tyear + 1)
        calnav['next']['month'] = '/1'

    if calnav['prev']['year'] == '/' + str(today.year):
        calnav['prev']['year'] = ''
        if calnav['prev']['month'] == '/' + str(today.month):
            calnav['prev']['month'] = ''

    if calnav['next']['year'] == '/' + str(today.year):
        calnav['next']['year'] = ''
        if calnav['next']['month'] == '/' + str(today.month):
            calnav['next']['month'] = ''

    calnavlinks = {
        'prev': calnav['prev']['year'] + calnav['prev']['month'],
        'next': calnav['next']['year'] + calnav['next']['month']
    }

    context = {'day_list': day_list, 'tyear': tyear, 'calnavlinks':
calnavlinks, 'monthname': monthnames[tmonth], 'tday': tday, 'currentmonth':
currentmonth}

    return render(request, 'lawncaresite/calendar.html', context)
```

```
@login_required
def one_job(request):

    jobs = get_jobs()
    job = None
    for j in jobs:
        if j.job_id == int(request.POST['job_id']):
            job = j

    dtime = {'Mow': 8, 'Fertilize': 2, 'Seed': 16, 'Brush': 40, 'Other': 10}

    duration = job.client.acreage * dtime[job.jtype]


    destination = job.client.address
    drive_data = get_drive_data(job.start, destination)

    drive_distance = math.floor(drive_data['distance'] / 1000)
    drive_time = timedelta(seconds=drive_data['time'])

    departure = datetime.combine(job.day, job.time) - drive_time
    departure = departure.time


    context = {'job': job, 'duration': duration, 'departure': departure,
'drive_distance': drive_distance, 'drive_time': drive_time}
    return render(request, 'lawncaresite/onejob.html', context)



def help(request):

    username = valid_user(request)

    context = {'username': username}

    return render(request, 'lawncaresite/help.html', context)



def index(request, message='none'):


    if request.user.is_authenticated:
        return redirect('calendar')

    form = LoginForm()

    context = {'form': form, 'message': message}
    return render(request, 'lawncaresite/index.html', context)



def login_user(request):
    login_form = LoginForm(request.POST)

    if login_form.is_valid():

        username = 'admin1'
        password = request.POST['password']

        user = authenticate(request, username=username,
            password=password)
        if user is not None:
            login(request, user)
            return redirect('calendar')

        else:
            return redirect('index', message = 'login failed')

    else:
        return redirect('index', message = 'invalid form')

    return redirect('index/', message = 'unexplained error')


def logout_user(request):
    logout(request)

    return redirect('index')



@login_required
def user(request, message='none'):

    password_form = ChangePassword()
    home_form = ChangeHome()

    home = HomeAddress.objects.all()[0].address;

    context = {'home': home, 'password_form': password_form, 'home_form':
home_form, 'message': message}

    return render(request, 'lawncaresite/user.html', context)
```

```
def change_password(request):
    form = ChangePassword(request.POST)
    if form.is_valid():

        checkusername = valid_user(request)

        user = authenticate(request, username=checkusername,
password=request.POST['currentpassword'])
        if user is not None:
            user.set_password(request.POST['newpassword'])
            user.save()
            login(request, user)
            request.session['username'] = checkusername
            return redirect('user', message = 'Password Successfully
Changed')
        return redirect('user', message = 'Incorrect Password')
    else:
        return redirect('user', message = 'invalid Input')
    return redirect('user', message = 'I dunno')



def change_home(request):
    form = ChangeHome(request.POST)
    if form.is_valid():

        full_address = request.POST['street'] + ', ' + request.POST['city'] +
', ' + request.POST['state']
        full_address2 = request.POST['street'] + ', ' + request.POST['city']

        home = HomeAddress.objects.all()[0].address;
        if get_drive_data('1 University Blvd St. Louis',
full_address)['valid'] and get_drive_data('1 University Blvd St. Louis',
full_address2)['valid']:

            home = HomeAddress.objects.all()[0]
            home.address = full_address
            home.save()
            return redirect('user')
        else:
            return redirect('user', message = 'Invalid Address')
    else:
        return redirect('user', message = 'Invalid Input')
```