INSTITUTE *of* TECHNOLOGY CARLOW

Institiúid Teicneolaíochta Cheatharlach

Name : Damien Doran

Student Number : C00221791

Project title : Teagasc app

Document : Technical Specification

Date : 2020/2021

Table of Contents:

Introduction 4
Table of Contents: 2
Table of Figuires: 3
Project Code 5
Views.py 5
Models.py 19
Tests 23
Forms.py 27
Templates 31

Table of Figures:

Figure 1: Views.py 5
Figure 2: Views.py 6
Figure 3: Views.py 7
Figure 4: Views.py 8
Figure 5: Views.py 9
Figure 6: Views.py 10
Figure 7: Views.py 11
Figure 8: Views.py 12
Figure 9: Views.py 13
Figure 10: Views.py 14
Figure 11: Views.py 15
Figure 12: Views.py 16
Figure 13: Views.py 17
Figure 14: Views.py 18
Figure 15: Models.py 19
Figure 16: Models.py 20
Figure 17: Models.py 21
Figure 18: Models.py 22
Figure 19: Test.py 23
Figure 20: Test.py 24
Figure 21: Test.py 25
Figure 22:Test.py 26
Figure 23: Forms.py 27
Figure 24: Forms. py 28
Figure 25: Forms.py 29
Figure 26: Forms.py 30
Figure 27: Templates.py 31
Figure 28: Templates.py 32
Figure 29: Templates.py 33
Figure 30: Templates.py 34

Introduction

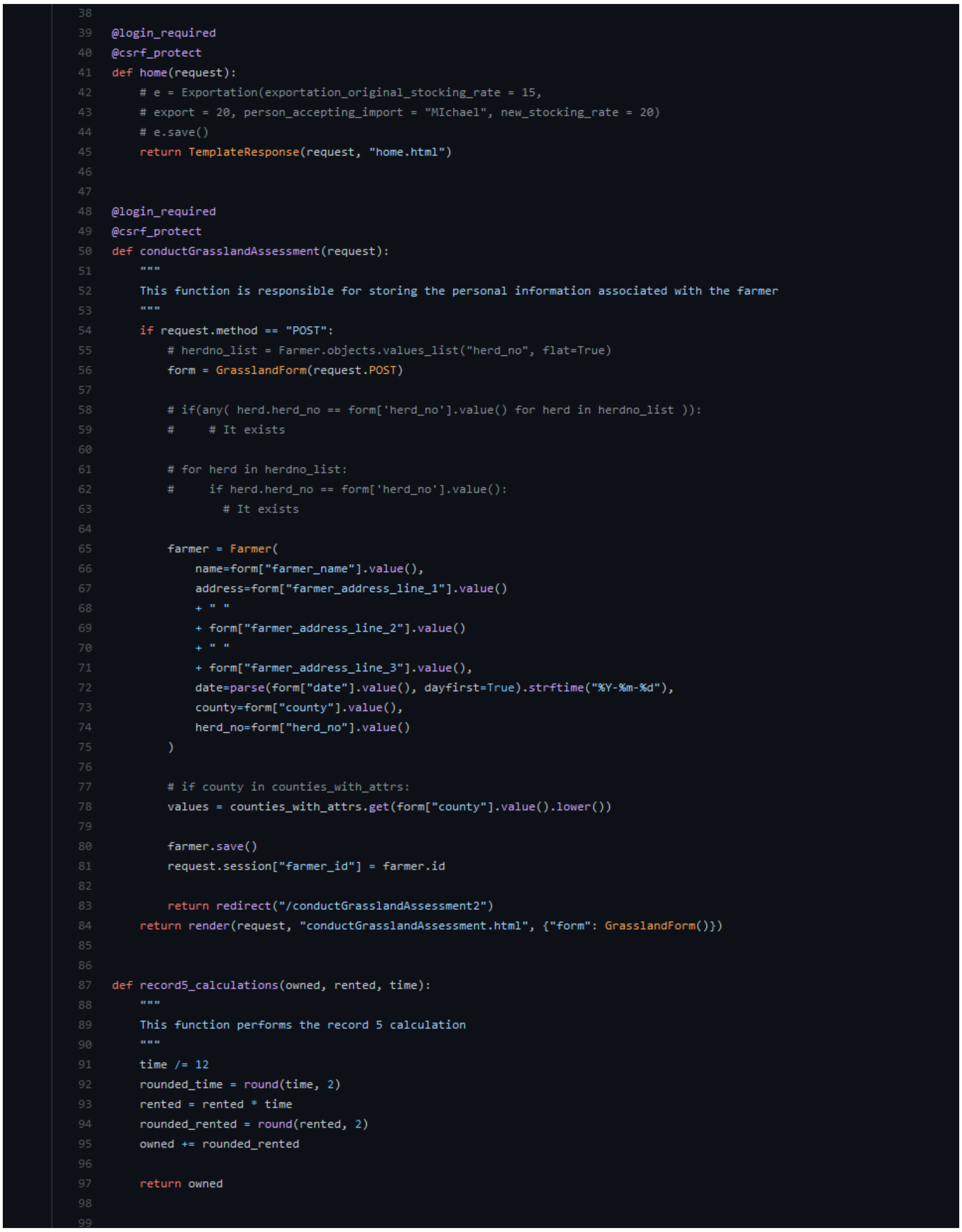
The purpose of this Technical Manual is to outline the requirements, installation procedure and show all the relevant code for the Teagasc Advisors application. The application will be deployed and available from http://teagascnmp.pythonanywhere.com/.

The code will be captured in screenshots and displayed throughout this document.

Project Code

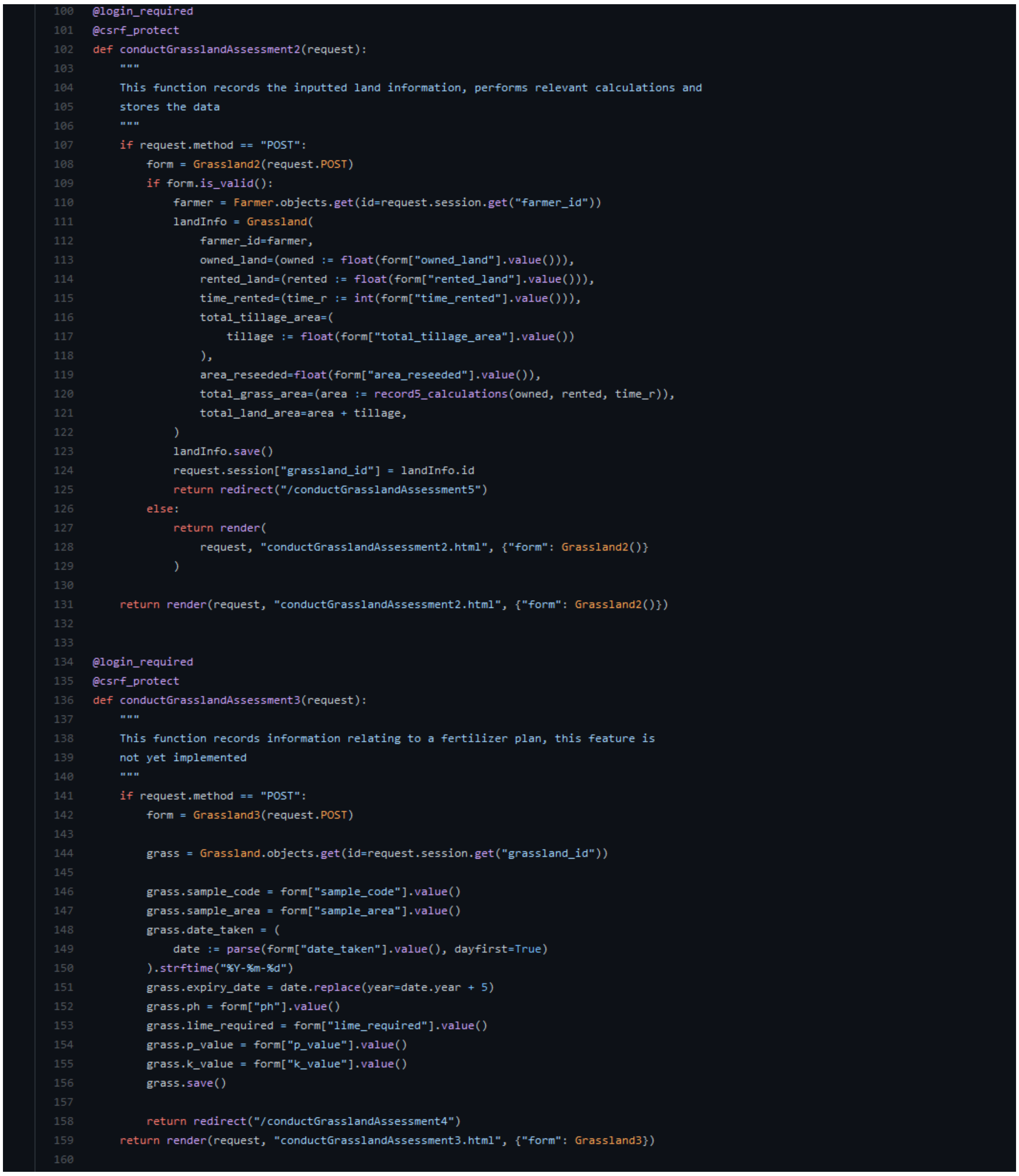
All the projects code is available on github at https://github.com/Suzukibud/Teagasc-App.

Views.py



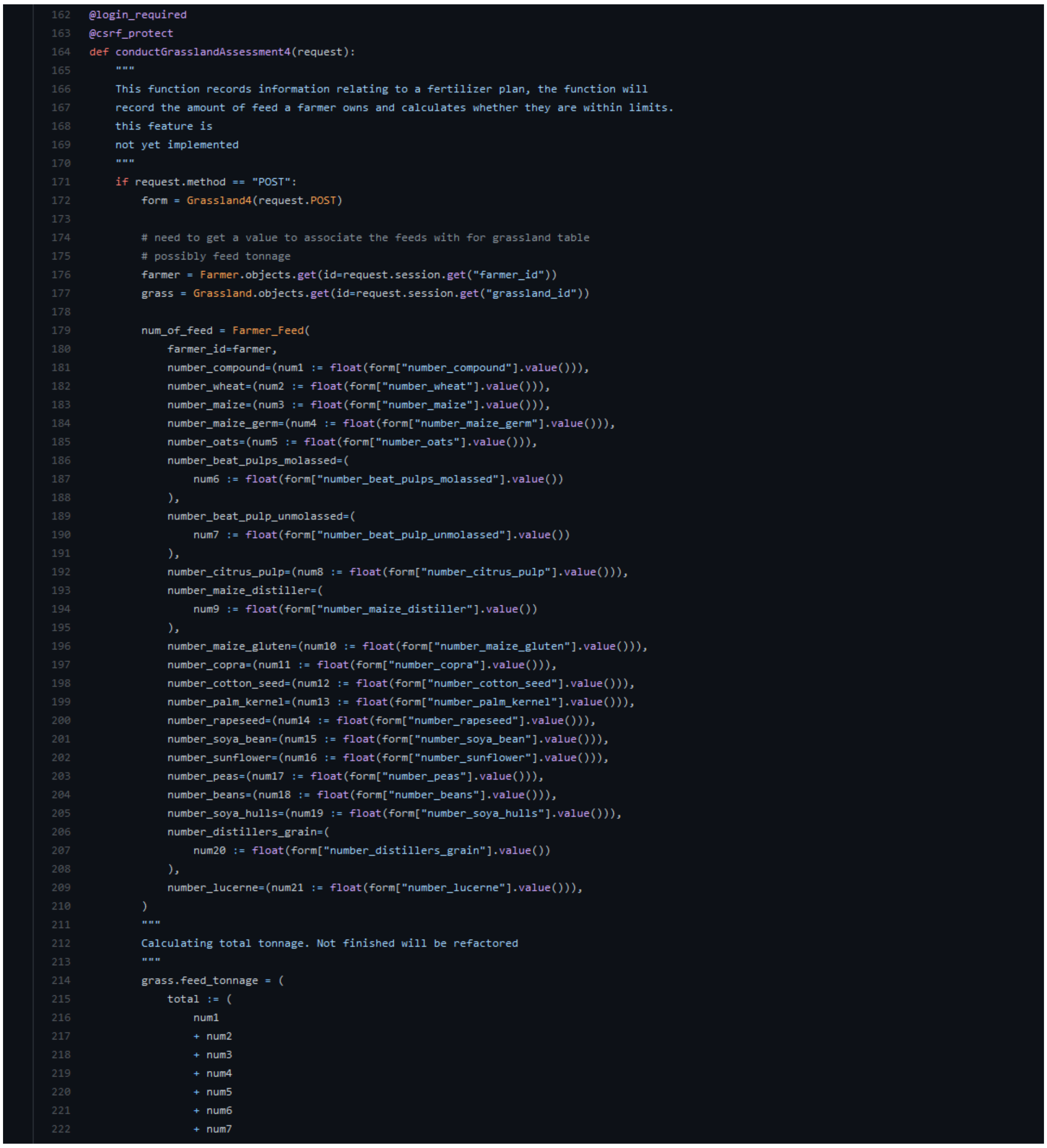
```python

@login_required
@csrf_protect
def home(request):
    # e = Exportation(exportation_original_stocking_rate = 15,
    # export = 20, person_accepting_import = "MIchael", new_stocking_rate = 20)
    # e.save()
    return TemplateResponse(request, "home.html")


@login_required
@csrf_protect
def conductGrasslandAssessment(request):
    """
    This function is responsible for storing the personal information associated with the farmer
    """
    if request.method == "POST":
        # herdno_list = Farmer.objects.values_list("herd_no", flat=True)
        form = GrasslandForm(request.POST)

        # if(any( herd.herd_no == form['herd_no'].value() for herd in herdno_list )):
        #     # It exists

        # for herd in herdno_list:
        #     if herd.herd_no == form['herd_no'].value():
                # It exists

        farmer = Farmer(
            name=form["farmer_name"].value(),
            address=form["farmer_address_line_1"].value()
            + " "
            + form["farmer_address_line_2"].value()
            + " "
            + form["farmer_address_line_3"].value(),
            date=parse(form["date"].value(), dayfirst=True).strftime("%Y-%m-%d"),
            county=form["county"].value(),
            herd_no=form["herd_no"].value()
        )

        # if county in counties_with_attrs:
        values = counties_with_attrs.get(form["county"].value().lower())

        farmer.save()
        request.session["farmer_id"] = farmer.id

        return redirect("/conductGrasslandAssessment2")
    return render(request, "conductGrasslandAssessment.html", {"form": GrasslandForm()})


def record5_calculations(owned, rented, time):
    """
    This function performs the record 5 calculation
    """
    time /= 12
    rounded_time = round(time, 2)
    rented = rented * time
    rounded_rented = round(rented, 2)
    owned += rounded_rented

    return owned


```

Figure 1: Views.py



```python
@login_required
@csrf_protect
def conductGrasslandAssessment2(request):
    """
    This function records the inputted land information, performs relevant calculations and
    stores the data
    """
    if request.method == "POST":
        form = Grassland2(request.POST)
        if form.is_valid():
            farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
            landInfo = Grassland(
                farmer_id=farmer,
                owned_land=(owned := float(form["owned_land"].value())),
                rented_land=(rented := float(form["rented_land"].value())),
                time_rented=(time_r := int(form["time_rented"].value())),
                total_tillage_area=(
                    tillage := float(form["total_tillage_area"].value())
                ),
                area_reseeded=float(form["area_reseeded"].value()),
                total_grass_area=(area := record5_calculations(owned, rented, time_r)),
                total_land_area=area + tillage,
            )
            landInfo.save()
            request.session["grassland_id"] = landInfo.id
            return redirect("/conductGrasslandAssessment5")
        else:
            return render(
                request, "conductGrasslandAssessment2.html", {"form": Grassland2()}
            )

    return render(request, "conductGrasslandAssessment2.html", {"form": Grassland2()})


@login_required
@csrf_protect
def conductGrasslandAssessment3(request):
    """
    This function records information relating to a fertilizer plan, this feature is
    not yet implemented
    """
    if request.method == "POST":
        form = Grassland3(request.POST)

        grass = Grassland.objects.get(id=request.session.get("grassland_id"))

        grass.sample_code = form["sample_code"].value()
        grass.sample_area = form["sample_area"].value()
        grass.date_taken = (
            date := parse(form["date_taken"].value(), dayfirst=True)
        ).strftime("%Y-%m-%d")
        grass.expiry_date = date.replace(year=date.year + 5)
        grass.ph = form["ph"].value()
        grass.lime_required = form["lime_required"].value()
        grass.p_value = form["p_value"].value()
        grass.k_value = form["k_value"].value()
        grass.save()

        return redirect("/conductGrasslandAssessment4")
    return render(request, "conductGrasslandAssessment3.html", {"form": Grassland3})
```

Figure 2: Views.py



```python
@login_required
@csrf_protect
def conductGrasslandAssessment4(request):
    """
    This function records information relating to a fertilizer plan, the function will
    record the amount of feed a farmer owns and calculates whether they are within limits.
    this feature is
    not yet implemented
    """
    if request.method == "POST":
        form = Grassland4(request.POST)

        # need to get a value to associate the feeds with for grassland table
        # possibly feed tonnage
        farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
        grass = Grassland.objects.get(id=request.session.get("grassland_id"))

        num_of_feed = Farmer_Feed(
            farmer_id=farmer,
            number_compound=(num1 := float(form["number_compound"].value())),
            number_wheat=(num2 := float(form["number_wheat"].value())),
            number_maize=(num3 := float(form["number_maize"].value())),
            number_maize_germ=(num4 := float(form["number_maize_germ"].value())),
            number_oats=(num5 := float(form["number_oats"].value())),
            number_beat_pulps_molassed=(
                num6 := float(form["number_beat_pulps_molassed"].value())
            ),
            number_beat_pulp_unmolassed=(
                num7 := float(form["number_beat_pulp_unmolassed"].value())
            ),
            number_citrus_pulp=(num8 := float(form["number_citrus_pulp"].value())),
            number_maize_distiller=(
                num9 := float(form["number_maize_distiller"].value())
            ),
            number_maize_gluten=(num10 := float(form["number_maize_gluten"].value())),
            number_copra=(num11 := float(form["number_copra"].value())),
            number_cotton_seed=(num12 := float(form["number_cotton_seed"].value())),
            number_palm_kernel=(num13 := float(form["number_palm_kernel"].value())),
            number_rapeseed=(num14 := float(form["number_rapeseed"].value())),
            number_soya_bean=(num15 := float(form["number_soya_bean"].value())),
            number_sunflower=(num16 := float(form["number_sunflower"].value())),
            number_peas=(num17 := float(form["number_peas"].value())),
            number_beans=(num18 := float(form["number_beans"].value())),
            number_soya_hulls=(num19 := float(form["number_soya_hulls"].value())),
            number_distillers_grain=(
                num20 := float(form["number_distillers_grain"].value())
            ),
            number_lucerne=(num21 := float(form["number_lucerne"].value())),
        )
        """
        Calculating total tonnage. Not finished will be refactored
        """
        grass.feed_tonnage = (
            total := (
                num1
                + num2
                + num3
                + num4
                + num5
                + num6
                + num7
```

Figure 3: Views.py

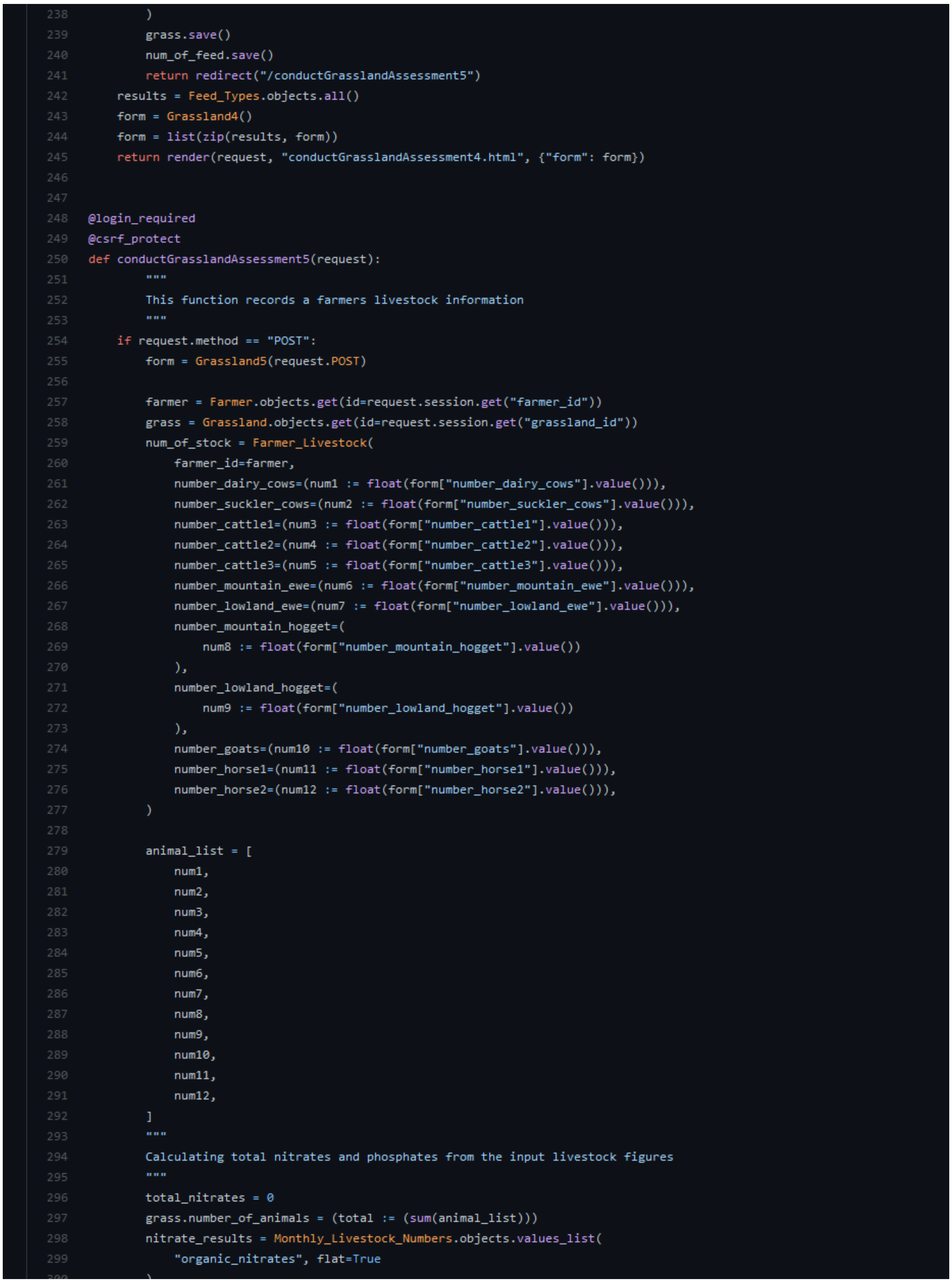
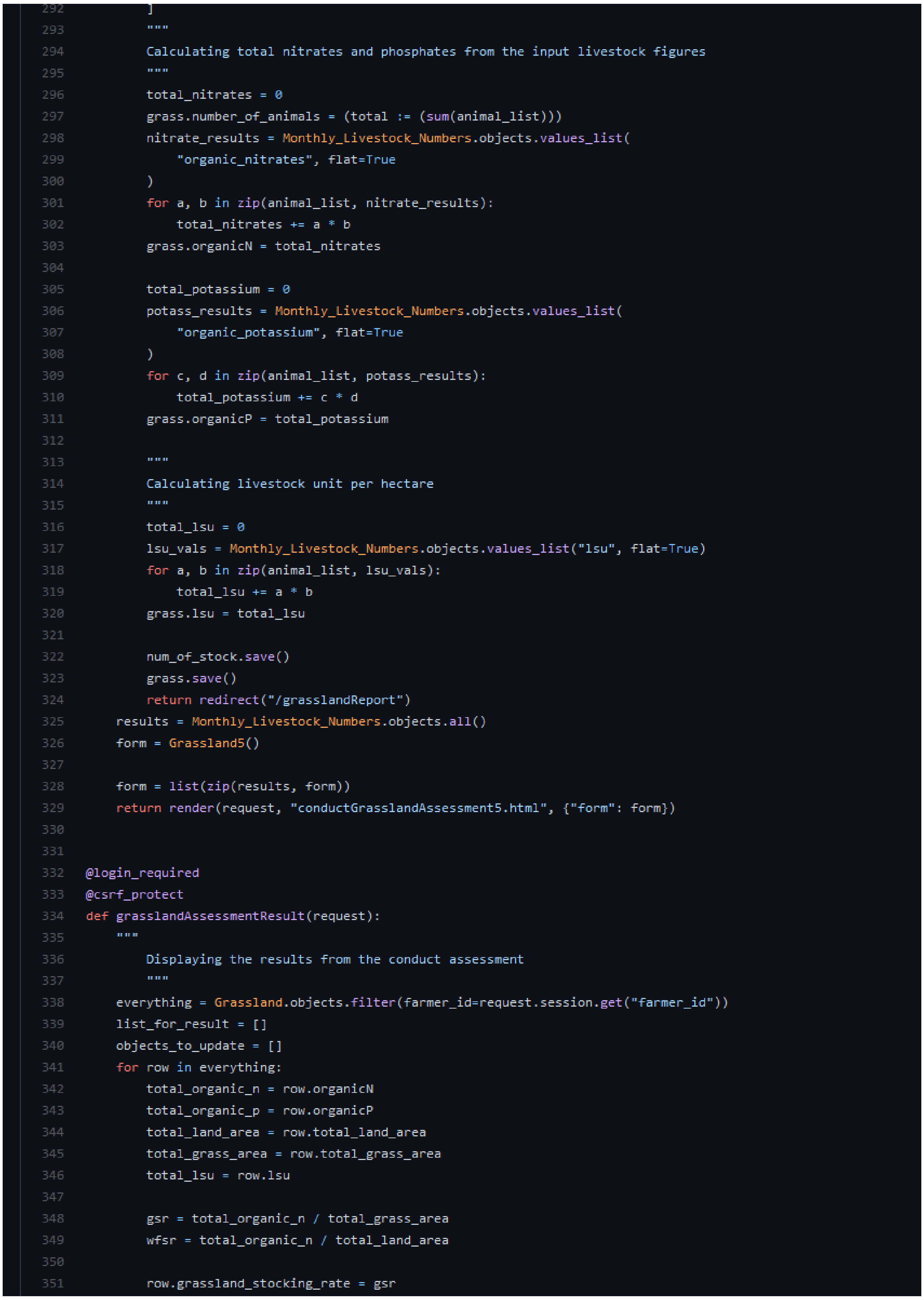
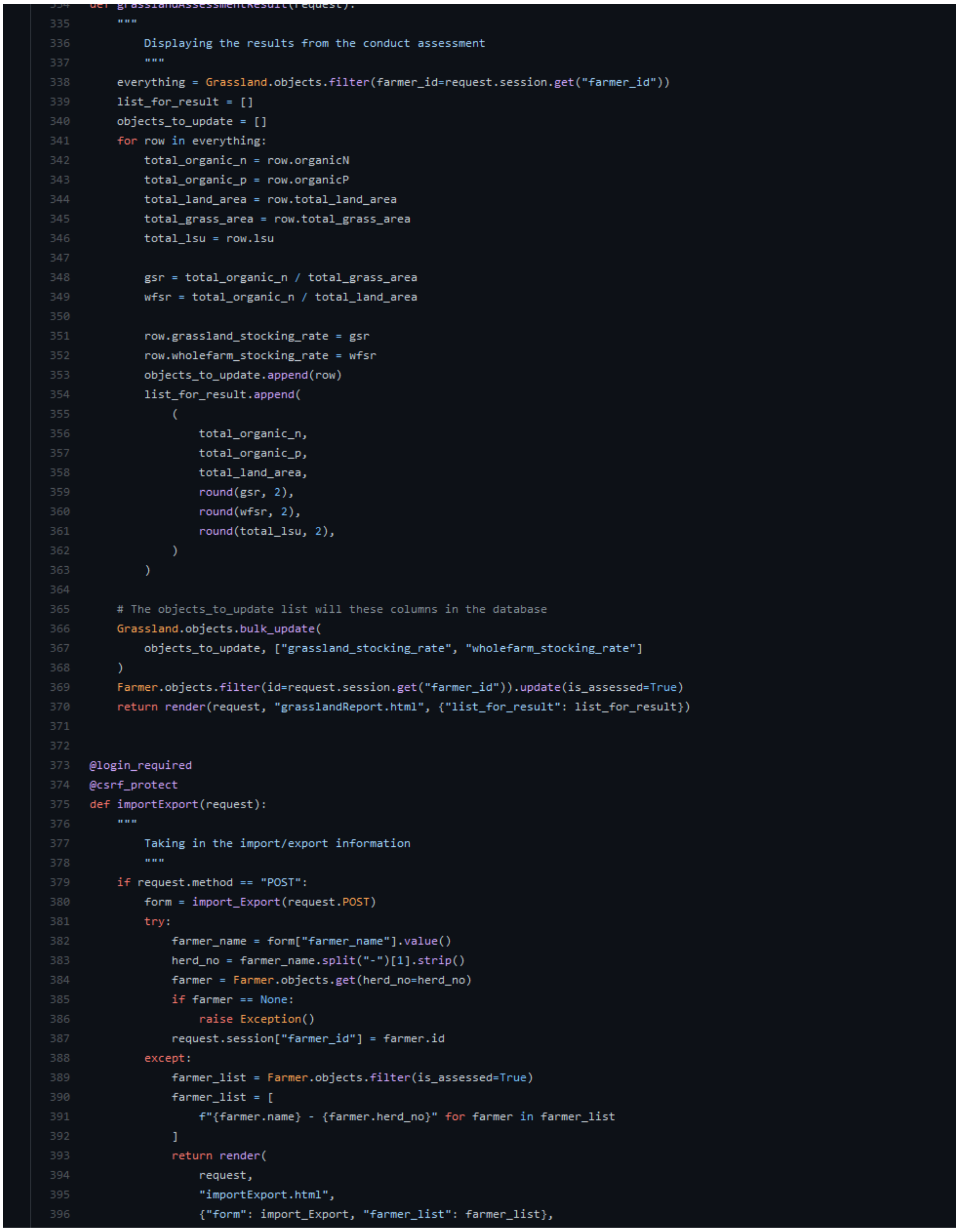

```python
        )
        grass.save()
        num_of_feed.save()
        return redirect("/conductGrasslandAssessment5")
    results = Feed_Types.objects.all()
    form = Grassland4()
    form = list(zip(results, form))
    return render(request, "conductGrasslandAssessment4.html", {"form": form})


@login_required
@csrf_protect
def conductGrasslandAssessment5(request):
        """
        This function records a farmers livestock information
        """
    if request.method == "POST":
        form = Grassland5(request.POST)

        farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
        grass = Grassland.objects.get(id=request.session.get("grassland_id"))
        num_of_stock = Farmer_Livestock(
            farmer_id=farmer,
            number_dairy_cows=(num1 := float(form["number_dairy_cows"].value())),
            number_suckler_cows=(num2 := float(form["number_suckler_cows"].value())),
            number_cattle1=(num3 := float(form["number_cattle1"].value())),
            number_cattle2=(num4 := float(form["number_cattle2"].value())),
            number_cattle3=(num5 := float(form["number_cattle3"].value())),
            number_mountain_ewe=(num6 := float(form["number_mountain_ewe"].value())),
            number_lowland_ewe=(num7 := float(form["number_lowland_ewe"].value())),
            number_mountain_hogget=(
                num8 := float(form["number_mountain_hogget"].value())
            ),
            number_lowland_hogget=(
                num9 := float(form["number_lowland_hogget"].value())
            ),
            number_goats=(num10 := float(form["number_goats"].value())),
            number_horse1=(num11 := float(form["number_horse1"].value())),
            number_horse2=(num12 := float(form["number_horse2"].value())),
        )

        animal_list = [
            num1,
            num2,
            num3,
            num4,
            num5,
            num6,
            num7,
            num8,
            num9,
            num10,
            num11,
            num12,
        ]
        """
        Calculating total nitrates and phosphates from the input livestock figures
        """
        total_nitrates = 0
        grass.number_of_animals = (total := (sum(animal_list)))
        nitrate_results = Monthly_Livestock_Numbers.objects.values_list(
            "organic_nitrates", flat=True
```

Figure 4: Views.py



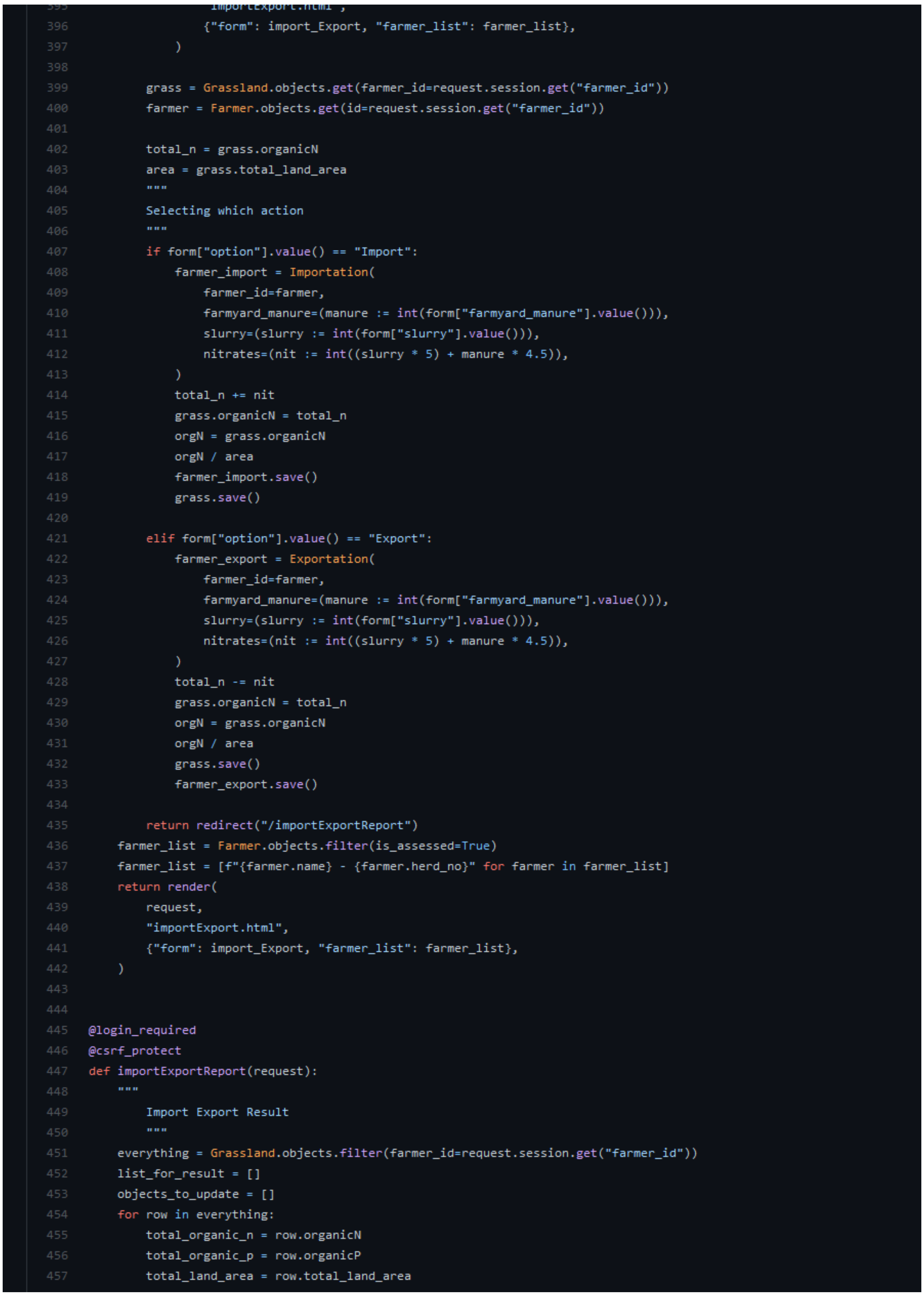
```
        ]
        """
        Calculating total nitrates and phosphates from the input livestock figures
        """
        total_nitrates = 0
        grass.number_of_animals = (total := (sum(animal_list)))
        nitrate_results = Monthly_Livestock_Numbers.objects.values_list(
            "organic_nitrates", flat=True
        )
        for a, b in zip(animal_list, nitrate_results):
            total_nitrates += a * b
        grass.organicN = total_nitrates

        total_potassium = 0
        potass_results = Monthly_Livestock_Numbers.objects.values_list(
            "organic_potassium", flat=True
        )
        for c, d in zip(animal_list, potass_results):
            total_potassium += c * d
        grass.organicP = total_potassium

        """
        Calculating livestock unit per hectare
        """
        total_lsu = 0
        lsu_vals = Monthly_Livestock_Numbers.objects.values_list("lsu", flat=True)
        for a, b in zip(animal_list, lsu_vals):
            total_lsu += a * b
        grass.lsu = total_lsu

        num_of_stock.save()
        grass.save()
        return redirect("/grasslandReport")
    results = Monthly_Livestock_Numbers.objects.all()
    form = Grassland5()

    form = list(zip(results, form))
    return render(request, "conductGrasslandAssessment5.html", {"form": form})


@login_required
@csrf_protect
def grasslandAssessmentResult(request):
    """
        Displaying the results from the conduct assessment
        """
    everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
    list_for_result = []
    objects_to_update = []
    for row in everything:
        total_organic_n = row.organicN
        total_organic_p = row.organicP
        total_land_area = row.total_land_area
        total_grass_area = row.total_grass_area
        total_lsu = row.lsu

        gsr = total_organic_n / total_grass_area
        wfsr = total_organic_n / total_land_area

        row.grassland_stocking_rate = gsr
```

Figure 5: Views.py



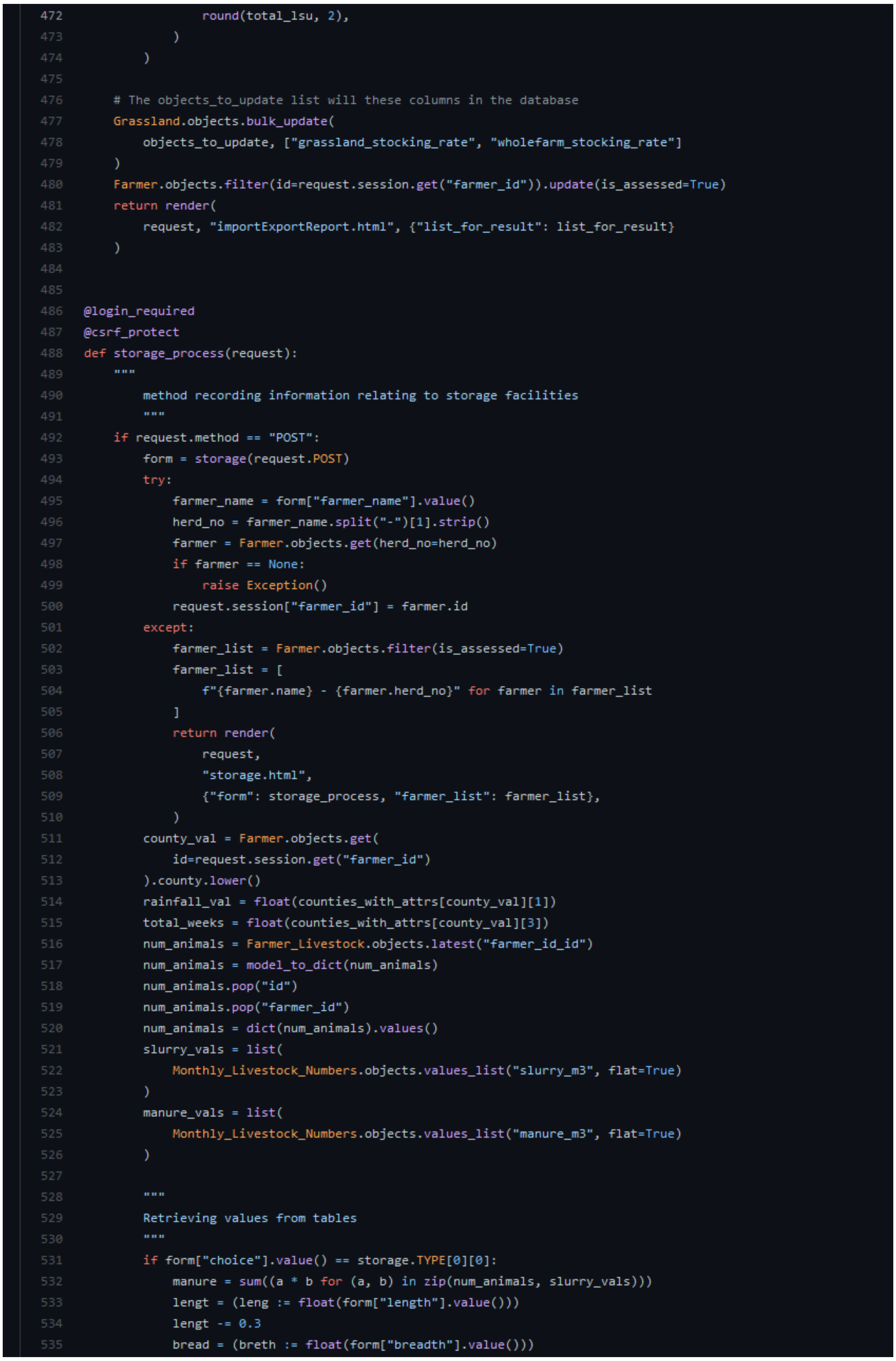
```python
def grasslandAssessmentResult(request):
    """
        Displaying the results from the conduct assessment
        """
    everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
    list_for_result = []
    objects_to_update = []
    for row in everything:
        total_organic_n = row.organicN
        total_organic_p = row.organicP
        total_land_area = row.total_land_area
        total_grass_area = row.total_grass_area
        total_lsu = row.lsu

        gsr = total_organic_n / total_grass_area
        wfsr = total_organic_n / total_land_area

        row.grassland_stocking_rate = gsr
        row.wholefarm_stocking_rate = wfsr
        objects_to_update.append(row)
        list_for_result.append(
            (
                total_organic_n,
                total_organic_p,
                total_land_area,
                round(gsr, 2),
                round(wfsr, 2),
                round(total_lsu, 2),
            )
        )

    # The objects_to_update list will these columns in the database
    Grassland.objects.bulk_update(
        objects_to_update, ["grassland_stocking_rate", "wholefarm_stocking_rate"]
    )
    Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
    return render(request, "grasslandReport.html", {"list_for_result": list_for_result})


@login_required
@csrf_protect
def importExport(request):
    """
        Taking in the import/export information
        """
    if request.method == "POST":
        form = import_Export(request.POST)
        try:
            farmer_name = form["farmer_name"].value()
            herd_no = farmer_name.split("-")[1].strip()
            farmer = Farmer.objects.get(herd_no=herd_no)
            if farmer == None:
                raise Exception()
            request.session["farmer_id"] = farmer.id
        except:
            farmer_list = Farmer.objects.filter(is_assessed=True)
            farmer_list = [
                f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
            ]
            return render(
                request,
                "importExport.html",
                {"form": import_Export, "farmer_list": farmer_list},
```

Figure 6: Views.py



```
                "importExport.html",
                {"form": import_Export, "farmer_list": farmer_list},
            )

        grass = Grassland.objects.get(farmer_id=request.session.get("farmer_id"))
        farmer = Farmer.objects.get(id=request.session.get("farmer_id"))

        total_n = grass.organicN
        area = grass.total_land_area
        """
        Selecting which action
        """
        if form["option"].value() == "Import":
            farmer_import = Importation(
                farmer_id=farmer,
                farmyard_manure=(manure := int(form["farmyard_manure"].value())),
                slurry=(slurry := int(form["slurry"].value())),
                nitrates=(nit := int((slurry * 5) + manure * 4.5)),
            )
            total_n += nit
            grass.organicN = total_n
            orgN = grass.organicN
            orgN / area
            farmer_import.save()
            grass.save()

        elif form["option"].value() == "Export":
            farmer_export = Exportation(
                farmer_id=farmer,
                farmyard_manure=(manure := int(form["farmyard_manure"].value())),
                slurry=(slurry := int(form["slurry"].value())),
                nitrates=(nit := int((slurry * 5) + manure * 4.5)),
            )
            total_n -= nit
            grass.organicN = total_n
            orgN = grass.organicN
            orgN / area
            grass.save()
            farmer_export.save()

        return redirect("/importExportReport")
    farmer_list = Farmer.objects.filter(is_assessed=True)
    farmer_list = [f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list]
    return render(
        request,
        "importExport.html",
        {"form": import_Export, "farmer_list": farmer_list},
    )


@login_required
@csrf_protect
def importExportReport(request):
    """
        Import Export Result
        """
    everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
    list_for_result = []
    objects_to_update = []
    for row in everything:
        total_organic_n = row.organicN
        total_organic_p = row.organicP
        total_land_area = row.total_land_area
```

Figure 7: Views.py



```python
                round(total_lsu, 2),
            )
        )

    # The objects_to_update list will these columns in the database
    Grassland.objects.bulk_update(
        objects_to_update, ["grassland_stocking_rate", "wholefarm_stocking_rate"]
    )
    Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
    return render(
        request, "importExportReport.html", {"list_for_result": list_for_result}
    )


@login_required
@csrf_protect
def storage_process(request):
    """
        method recording information relating to storage facilities
        """
    if request.method == "POST":
        form = storage(request.POST)
        try:
            farmer_name = form["farmer_name"].value()
            herd_no = farmer_name.split("-")[1].strip()
            farmer = Farmer.objects.get(herd_no=herd_no)
            if farmer == None:
                raise Exception()
            request.session["farmer_id"] = farmer.id
        except:
            farmer_list = Farmer.objects.filter(is_assessed=True)
            farmer_list = [
                f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
            ]
            return render(
                request,
                "storage.html",
                {"form": storage_process, "farmer_list": farmer_list},
            )
        county_val = Farmer.objects.get(
            id=request.session.get("farmer_id")
        ).county.lower()
        rainfall_val = float(counties_with_attrs[county_val][1])
        total_weeks = float(counties_with_attrs[county_val][3])
        num_animals = Farmer_Livestock.objects.latest("farmer_id_id")
        num_animals = model_to_dict(num_animals)
        num_animals.pop("id")
        num_animals.pop("farmer_id")
        num_animals = dict(num_animals).values()
        slurry_vals = list(
            Monthly_Livestock_Numbers.objects.values_list("slurry_m3", flat=True)
        )
        manure_vals = list(
            Monthly_Livestock_Numbers.objects.values_list("manure_m3", flat=True)
        )

        """
        Retrieving values from tables
        """
        if form["choice"].value() == storage.TYPE[0][0]:
            manure = sum((a * b for (a, b) in zip(num_animals, slurry_vals)))
            lengt = (leng := float(form["length"].value()))
            lengt -= 0.3
            bread = (breth := float(form["breadth"].value()))
```

Figure 8: Views.py

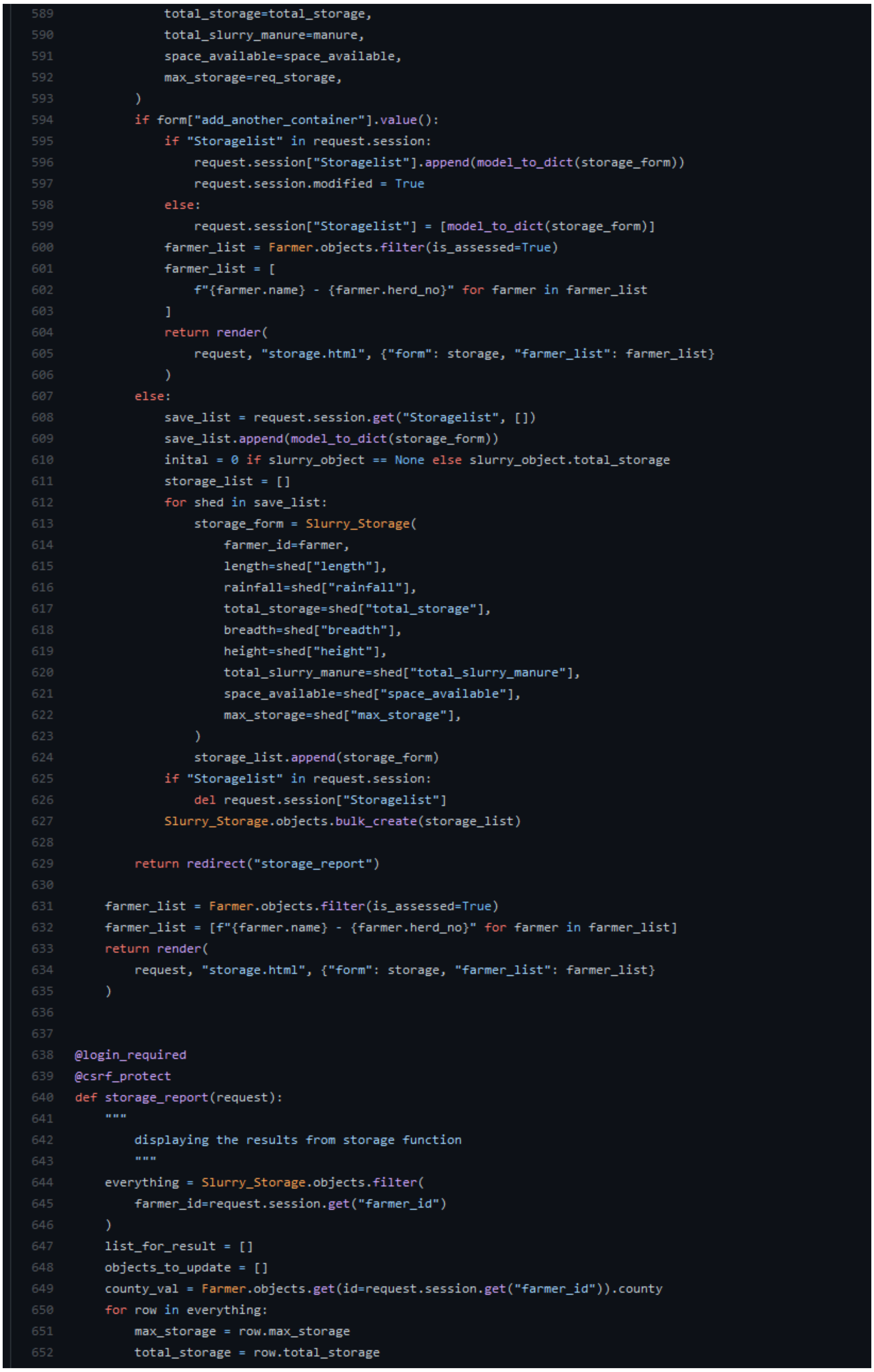
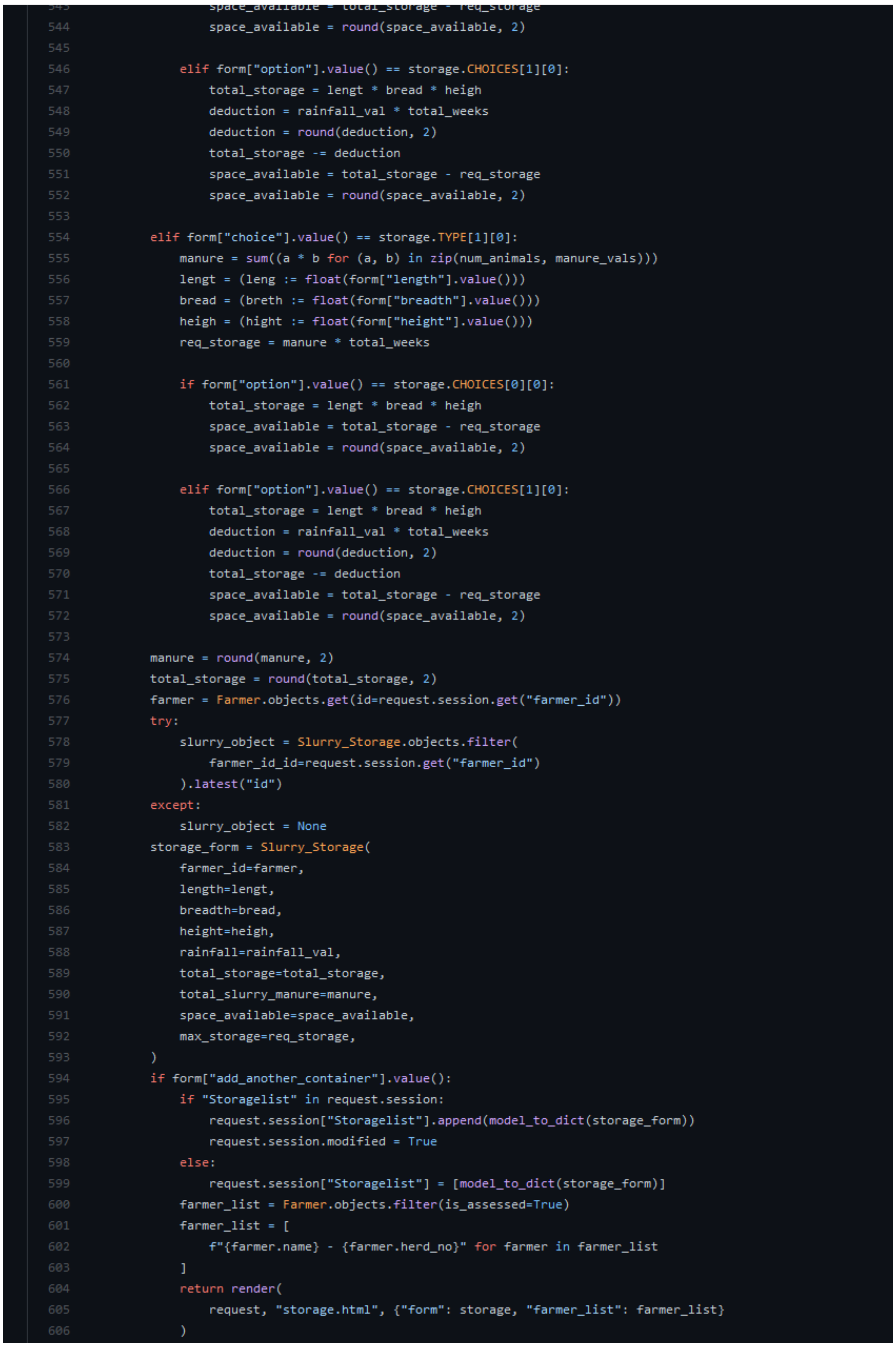
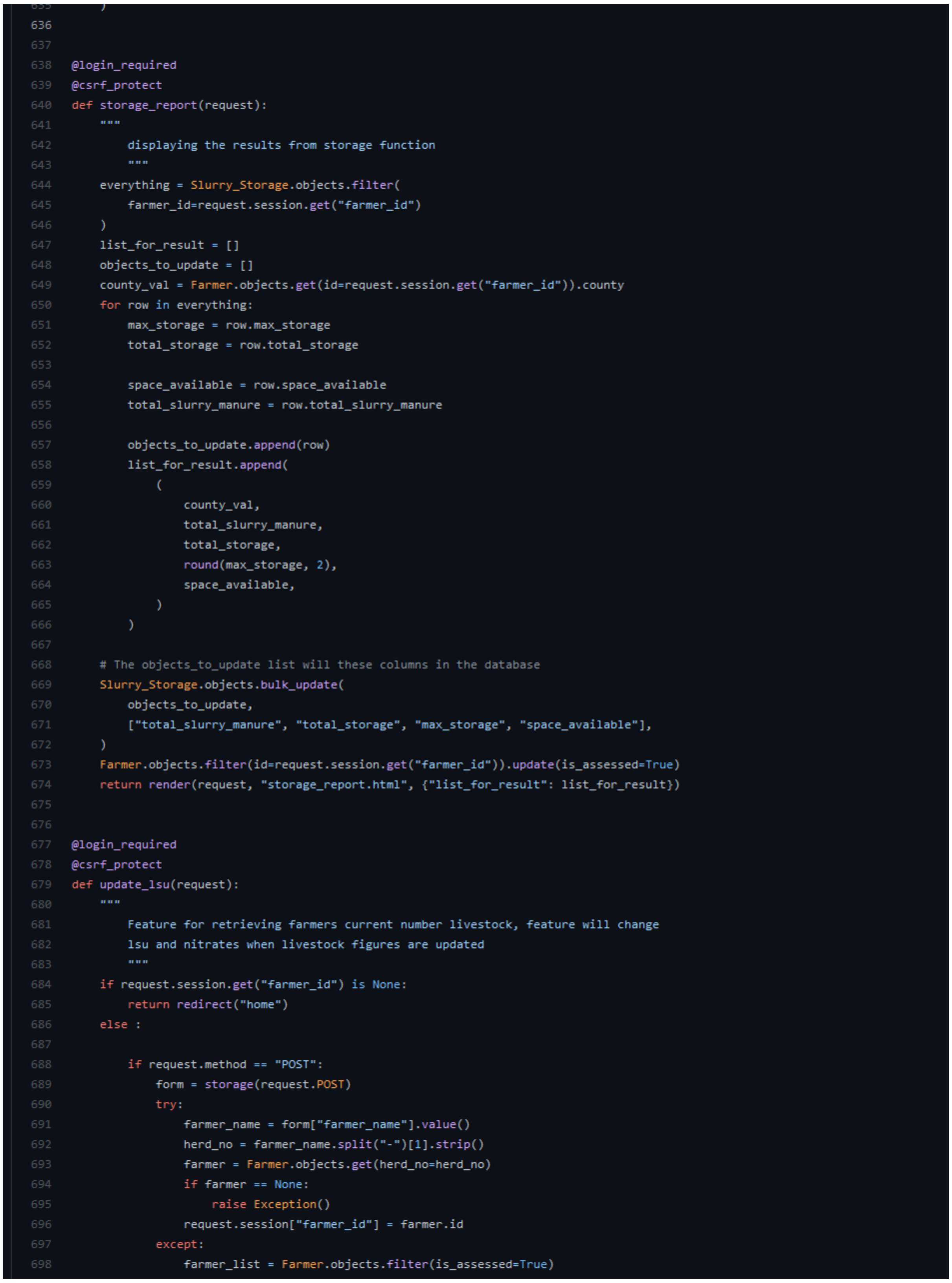

```python
            total_storage=total_storage,
            total_slurry_manure=manure,
            space_available=space_available,
            max_storage=req_storage,
        )
        if form["add_another_container"].value():
            if "Storagelist" in request.session:
                request.session["Storagelist"].append(model_to_dict(storage_form))
                request.session.modified = True
            else:
                request.session["Storagelist"] = [model_to_dict(storage_form)]
            farmer_list = Farmer.objects.filter(is_assessed=True)
            farmer_list = [
                f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
            ]
            return render(
                request, "storage.html", {"form": storage, "farmer_list": farmer_list}
            )
        else:
            save_list = request.session.get("Storagelist", [])
            save_list.append(model_to_dict(storage_form))
            inital = 0 if slurry_object == None else slurry_object.total_storage
            storage_list = []
            for shed in save_list:
                storage_form = Slurry_Storage(
                    farmer_id=farmer,
                    length=shed["length"],
                    rainfall=shed["rainfall"],
                    total_storage=shed["total_storage"],
                    breadth=shed["breadth"],
                    height=shed["height"],
                    total_slurry_manure=shed["total_slurry_manure"],
                    space_available=shed["space_available"],
                    max_storage=shed["max_storage"],
                )
                storage_list.append(storage_form)
            if "Storagelist" in request.session:
                del request.session["Storagelist"]
            Slurry_Storage.objects.bulk_create(storage_list)

        return redirect("storage_report")

    farmer_list = Farmer.objects.filter(is_assessed=True)
    farmer_list = [f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list]
    return render(
        request, "storage.html", {"form": storage, "farmer_list": farmer_list}
    )


@login_required
@csrf_protect
def storage_report(request):
    """
        displaying the results from storage function
        """
    everything = Slurry_Storage.objects.filter(
        farmer_id=request.session.get("farmer_id")
    )
    list_for_result = []
    objects_to_update = []
    county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
    for row in everything:
        max_storage = row.max_storage
        total_storage = row.total_storage
```

Figure 9: Views.py



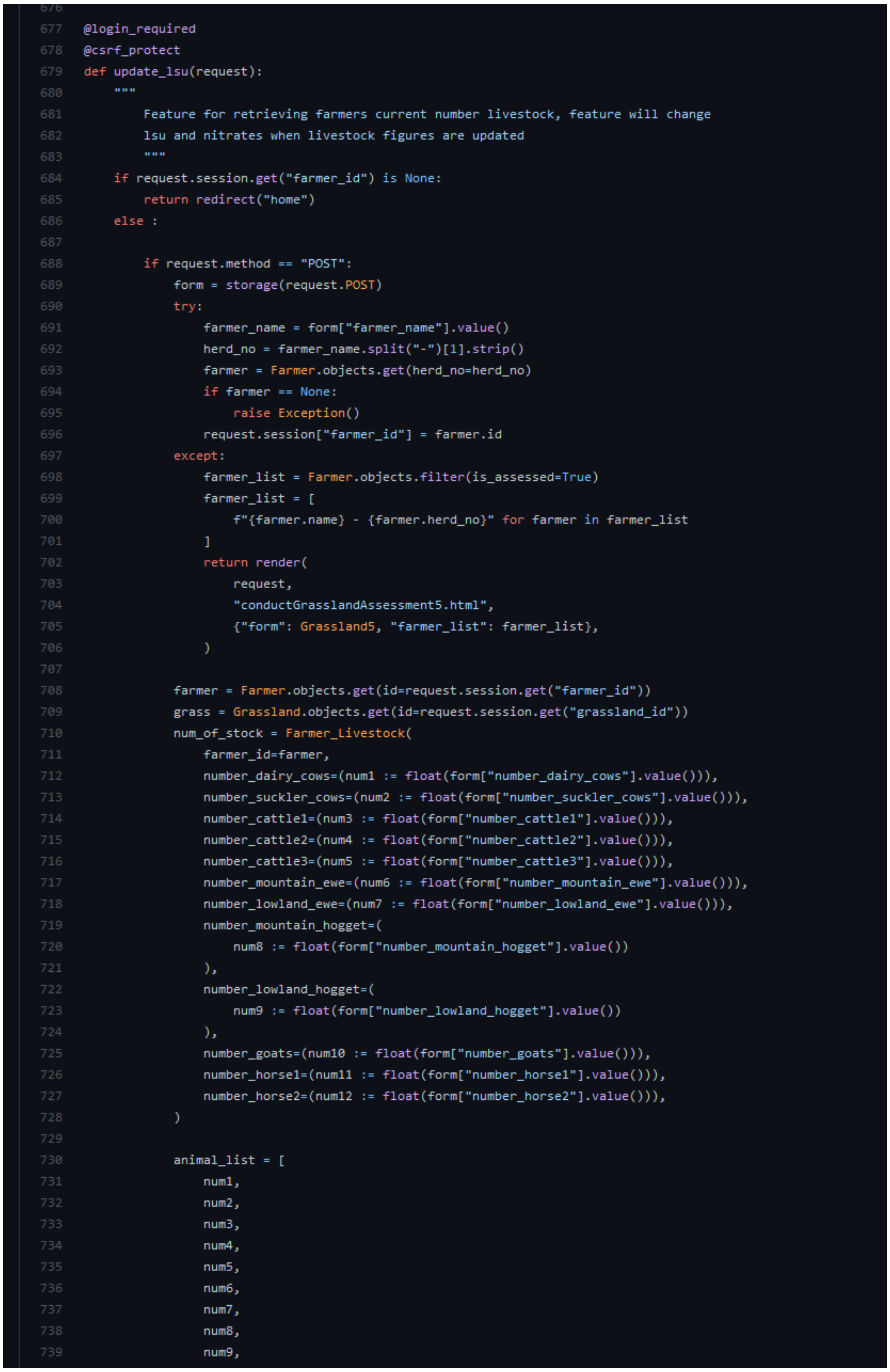
```python
                space_available = total_storage - req_storage
                space_available = round(space_available, 2)

            elif form["option"].value() == storage.CHOICES[1][0]:
                total_storage = lengt * bread * heigh
                deduction = rainfall_val * total_weeks
                deduction = round(deduction, 2)
                total_storage -= deduction
                space_available = total_storage - req_storage
                space_available = round(space_available, 2)

        elif form["choice"].value() == storage.TYPE[1][0]:
            manure = sum((a * b for (a, b) in zip(num_animals, manure_vals)))
            lengt = (leng := float(form["length"].value()))
            bread = (breth := float(form["breadth"].value()))
            heigh = (hight := float(form["height"].value()))
            req_storage = manure * total_weeks

            if form["option"].value() == storage.CHOICES[0][0]:
                total_storage = lengt * bread * heigh
                space_available = total_storage - req_storage
                space_available = round(space_available, 2)

            elif form["option"].value() == storage.CHOICES[1][0]:
                total_storage = lengt * bread * heigh
                deduction = rainfall_val * total_weeks
                deduction = round(deduction, 2)
                total_storage -= deduction
                space_available = total_storage - req_storage
                space_available = round(space_available, 2)

        manure = round(manure, 2)
        total_storage = round(total_storage, 2)
        farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
        try:
            slurry_object = Slurry_Storage.objects.filter(
                farmer_id_id=request.session.get("farmer_id")
            ).latest("id")
        except:
            slurry_object = None
        storage_form = Slurry_Storage(
            farmer_id=farmer,
            length=lengt,
            breadth=bread,
            height=heigh,
            rainfall=rainfall_val,
            total_storage=total_storage,
            total_slurry_manure=manure,
            space_available=space_available,
            max_storage=req_storage,
        )
        if form["add_another_container"].value():
            if "Storagelist" in request.session:
                request.session["Storagelist"].append(model_to_dict(storage_form))
                request.session.modified = True
            else:
                request.session["Storagelist"] = [model_to_dict(storage_form)]
            farmer_list = Farmer.objects.filter(is_assessed=True)
            farmer_list = [
                f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
            ]
            return render(
                request, "storage.html", {"form": storage, "farmer_list": farmer_list}
            )
```

Figure 10: Views.py



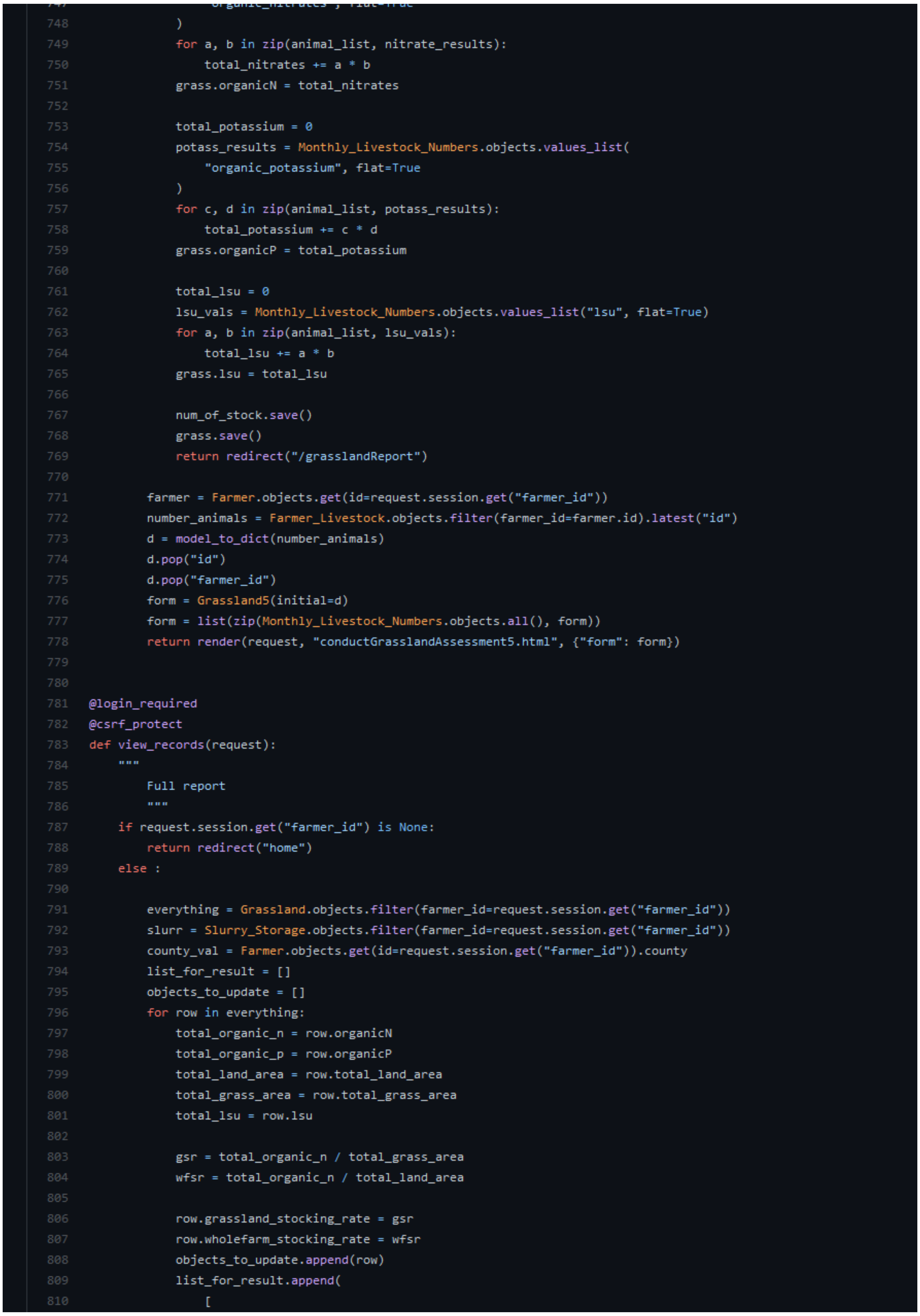
```
    )


@login_required
@csrf_protect
def storage_report(request):
    """
        displaying the results from storage function
        """
    everything = Slurry_Storage.objects.filter(
        farmer_id=request.session.get("farmer_id")
    )
    list_for_result = []
    objects_to_update = []
    county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
    for row in everything:
        max_storage = row.max_storage
        total_storage = row.total_storage

        space_available = row.space_available
        total_slurry_manure = row.total_slurry_manure

        objects_to_update.append(row)
        list_for_result.append(
            (
                county_val,
                total_slurry_manure,
                total_storage,
                round(max_storage, 2),
                space_available,
            )
        )

    # The objects_to_update list will these columns in the database
    Slurry_Storage.objects.bulk_update(
        objects_to_update,
        ["total_slurry_manure", "total_storage", "max_storage", "space_available"],
    )
    Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
    return render(request, "storage_report.html", {"list_for_result": list_for_result})


@login_required
@csrf_protect
def update_lsu(request):
    """
        Feature for retrieving farmers current number livestock, feature will change
        lsu and nitrates when livestock figures are updated
        """
    if request.session.get("farmer_id") is None:
        return redirect("home")
    else :

        if request.method == "POST":
            form = storage(request.POST)
            try:
                farmer_name = form["farmer_name"].value()
                herd_no = farmer_name.split("-")[1].strip()
                farmer = Farmer.objects.get(herd_no=herd_no)
                if farmer == None:
                    raise Exception()
                request.session["farmer_id"] = farmer.id
            except:
                farmer_list = Farmer.objects.filter(is_assessed=True)
```

Figure 11: Views.py



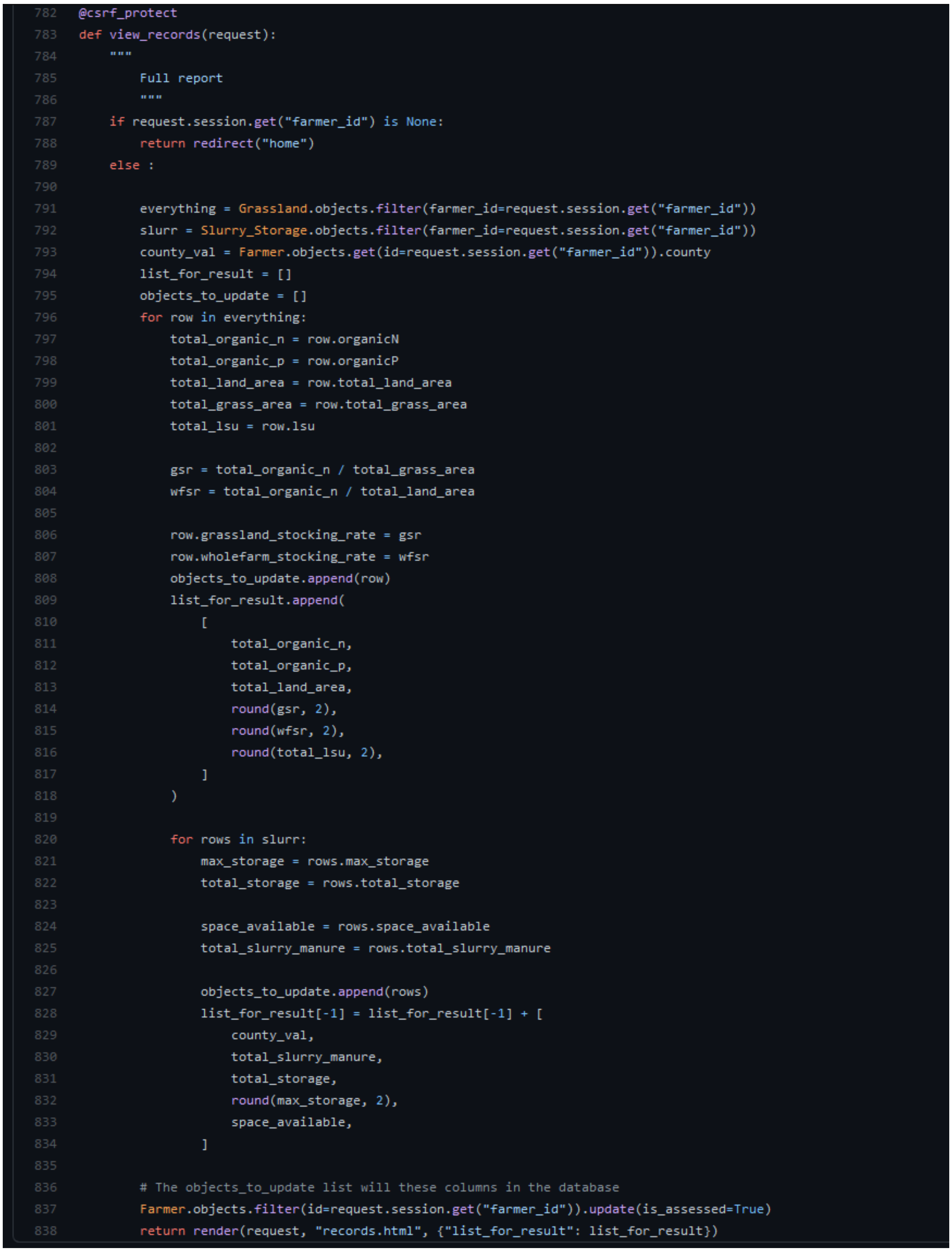
```python
@login_required
@csrf_protect
def update_lsu(request):
    """
        Feature for retrieving farmers current number livestock, feature will change
        lsu and nitrates when livestock figures are updated
        """
    if request.session.get("farmer_id") is None:
        return redirect("home")
    else :

        if request.method == "POST":
            form = storage(request.POST)
            try:
                farmer_name = form["farmer_name"].value()
                herd_no = farmer_name.split("-")[1].strip()
                farmer = Farmer.objects.get(herd_no=herd_no)
                if farmer == None:
                    raise Exception()
                request.session["farmer_id"] = farmer.id
            except:
                farmer_list = Farmer.objects.filter(is_assessed=True)
                farmer_list = [
                    f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
                ]
                return render(
                    request,
                    "conductGrasslandAssessment5.html",
                    {"form": Grassland5, "farmer_list": farmer_list},
                )

            farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
            grass = Grassland.objects.get(id=request.session.get("grassland_id"))
            num_of_stock = Farmer_Livestock(
                farmer_id=farmer,
                number_dairy_cows=(num1 := float(form["number_dairy_cows"].value())),
                number_suckler_cows=(num2 := float(form["number_suckler_cows"].value())),
                number_cattle1=(num3 := float(form["number_cattle1"].value())),
                number_cattle2=(num4 := float(form["number_cattle2"].value())),
                number_cattle3=(num5 := float(form["number_cattle3"].value())),
                number_mountain_ewe=(num6 := float(form["number_mountain_ewe"].value())),
                number_lowland_ewe=(num7 := float(form["number_lowland_ewe"].value())),
                number_mountain_hogget=(
                    num8 := float(form["number_mountain_hogget"].value())
                ),
                number_lowland_hogget=(
                    num9 := float(form["number_lowland_hogget"].value())
                ),
                number_goats=(num10 := float(form["number_goats"].value())),
                number_horse1=(num11 := float(form["number_horse1"].value())),
                number_horse2=(num12 := float(form["number_horse2"].value())),
            )

            animal_list = [
                num1,
                num2,
                num3,
                num4,
                num5,
                num6,
                num7,
                num8,
                num9,
```

Figure 12: Views.py



```python
                "organic_nitrates", flat=True
            )
            for a, b in zip(animal_list, nitrate_results):
                total_nitrates += a * b
            grass.organicN = total_nitrates

            total_potassium = 0
            potass_results = Monthly_Livestock_Numbers.objects.values_list(
                "organic_potassium", flat=True
            )
            for c, d in zip(animal_list, potass_results):
                total_potassium += c * d
            grass.organicP = total_potassium

            total_lsu = 0
            lsu_vals = Monthly_Livestock_Numbers.objects.values_list("lsu", flat=True)
            for a, b in zip(animal_list, lsu_vals):
                total_lsu += a * b
            grass.lsu = total_lsu

            num_of_stock.save()
            grass.save()
            return redirect("/grasslandReport")

        farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
        number_animals = Farmer_Livestock.objects.filter(farmer_id=farmer.id).latest("id")
        d = model_to_dict(number_animals)
        d.pop("id")
        d.pop("farmer_id")
        form = Grassland5(initial=d)
        form = list(zip(Monthly_Livestock_Numbers.objects.all(), form))
        return render(request, "conductGrasslandAssessment5.html", {"form": form})


@login_required
@csrf_protect
def view_records(request):
    """
        Full report
        """
    if request.session.get("farmer_id") is None:
        return redirect("home")
    else :

        everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
        slurr = Slurry_Storage.objects.filter(farmer_id=request.session.get("farmer_id"))
        county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
        list_for_result = []
        objects_to_update = []
        for row in everything:
            total_organic_n = row.organicN
            total_organic_p = row.organicP
            total_land_area = row.total_land_area
            total_grass_area = row.total_grass_area
            total_lsu = row.lsu

            gsr = total_organic_n / total_grass_area
            wfsr = total_organic_n / total_land_area

            row.grassland_stocking_rate = gsr
            row.wholefarm_stocking_rate = wfsr
            objects_to_update.append(row)
            list_for_result.append(
                [
```

Figure 13: Views.py



```python
@csrf_protect
def view_records(request):
    """
        Full report
        """
    if request.session.get("farmer_id") is None:
        return redirect("home")
    else :

        everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
        slurr = Slurry_Storage.objects.filter(farmer_id=request.session.get("farmer_id"))
        county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
        list_for_result = []
        objects_to_update = []
        for row in everything:
            total_organic_n = row.organicN
            total_organic_p = row.organicP
            total_land_area = row.total_land_area
            total_grass_area = row.total_grass_area
            total_lsu = row.lsu

            gsr = total_organic_n / total_grass_area
            wfsr = total_organic_n / total_land_area

            row.grassland_stocking_rate = gsr
            row.wholefarm_stocking_rate = wfsr
            objects_to_update.append(row)
            list_for_result.append(
                [
                    total_organic_n,
                    total_organic_p,
                    total_land_area,
                    round(gsr, 2),
                    round(wfsr, 2),
                    round(total_lsu, 2),
                ]
            )

            for rows in slurr:
                max_storage = rows.max_storage
                total_storage = rows.total_storage

                space_available = rows.space_available
                total_slurry_manure = rows.total_slurry_manure

                objects_to_update.append(rows)
                list_for_result[-1] = list_for_result[-1] + [
                    county_val,
                    total_slurry_manure,
                    total_storage,
                    round(max_storage, 2),
                    space_available,
                ]

        # The objects_to_update list will these columns in the database
        Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
        return render(request, "records.html", {"list_for_result": list_for_result})
```

Figure 14: Views.py

Models.py

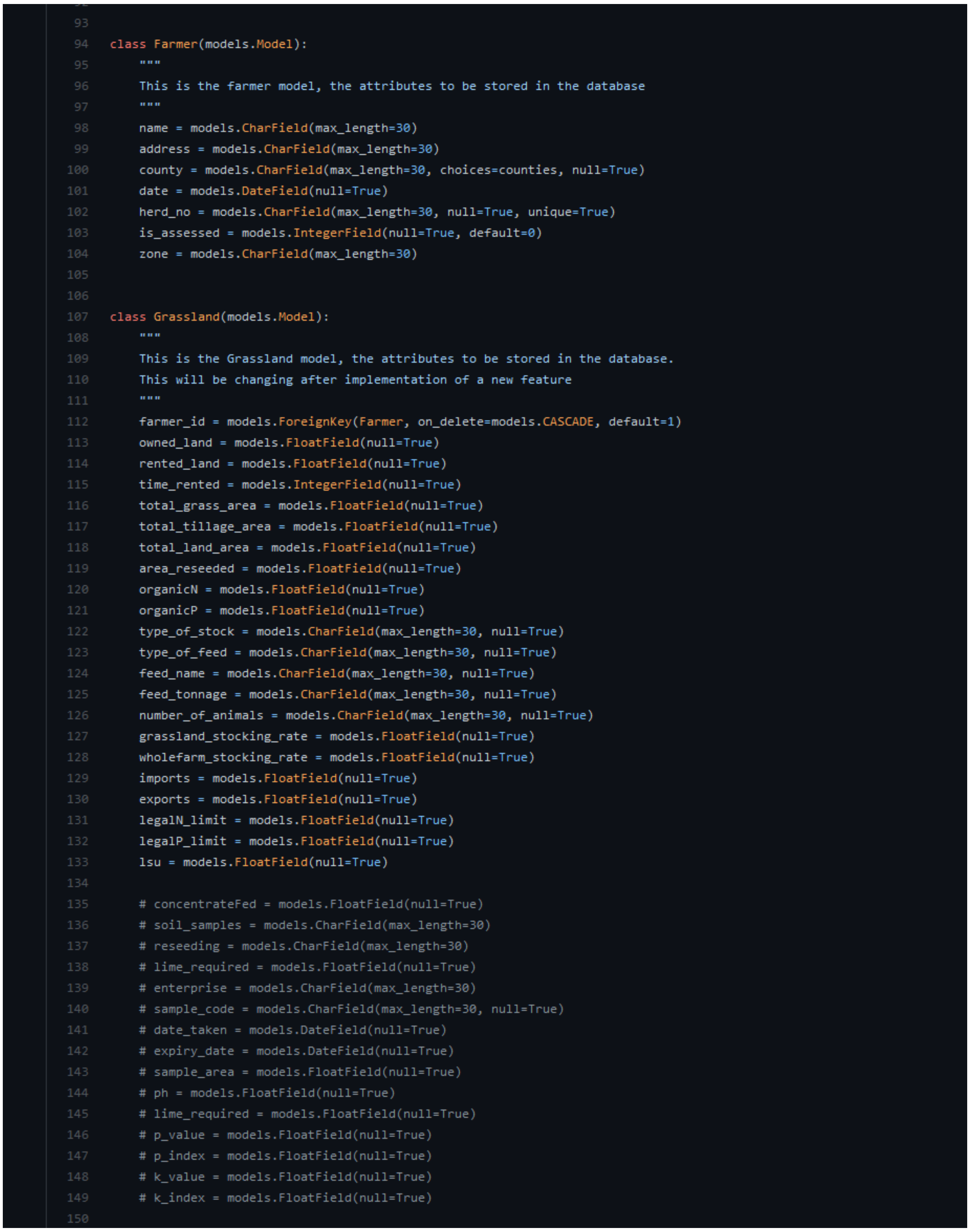
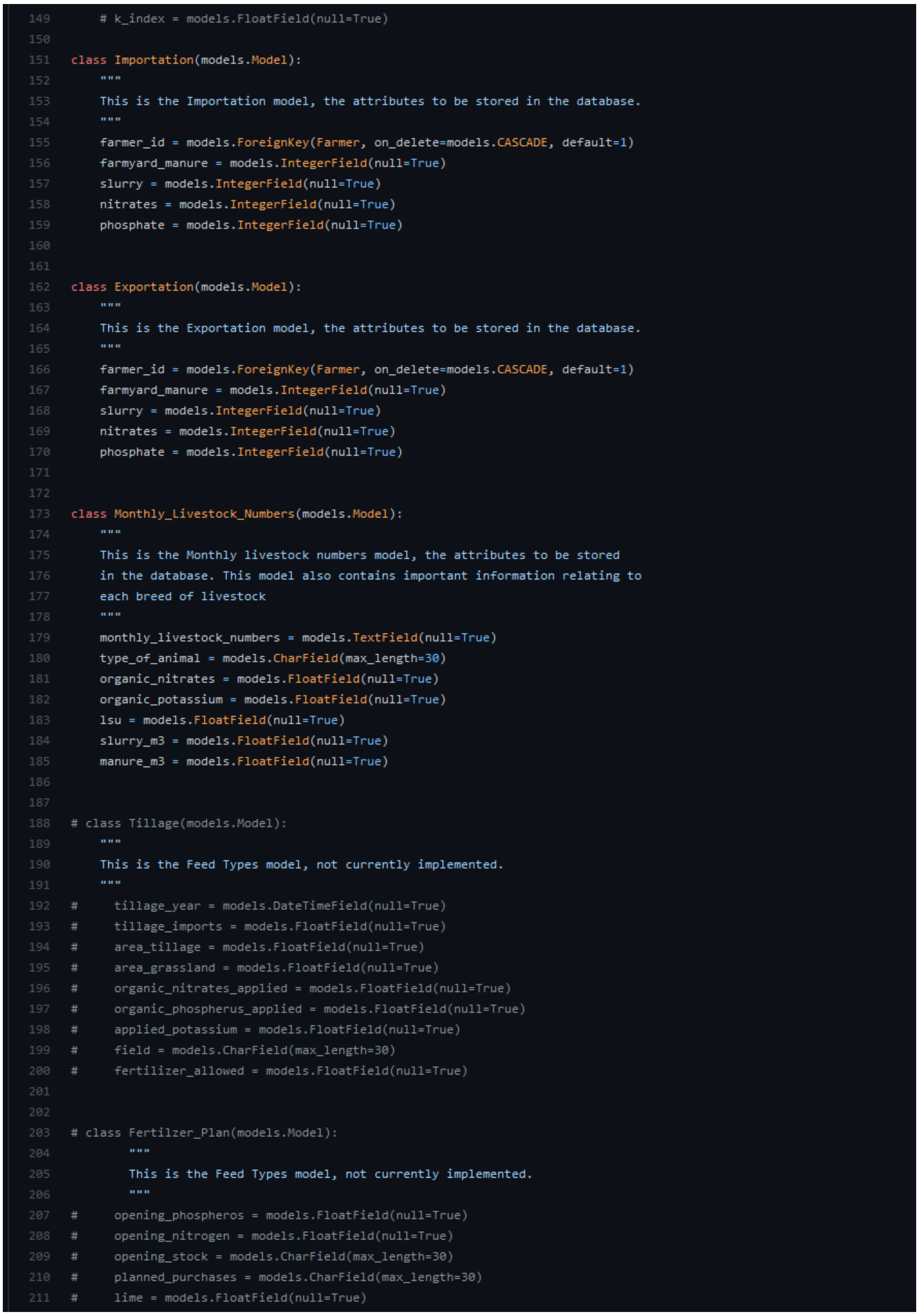
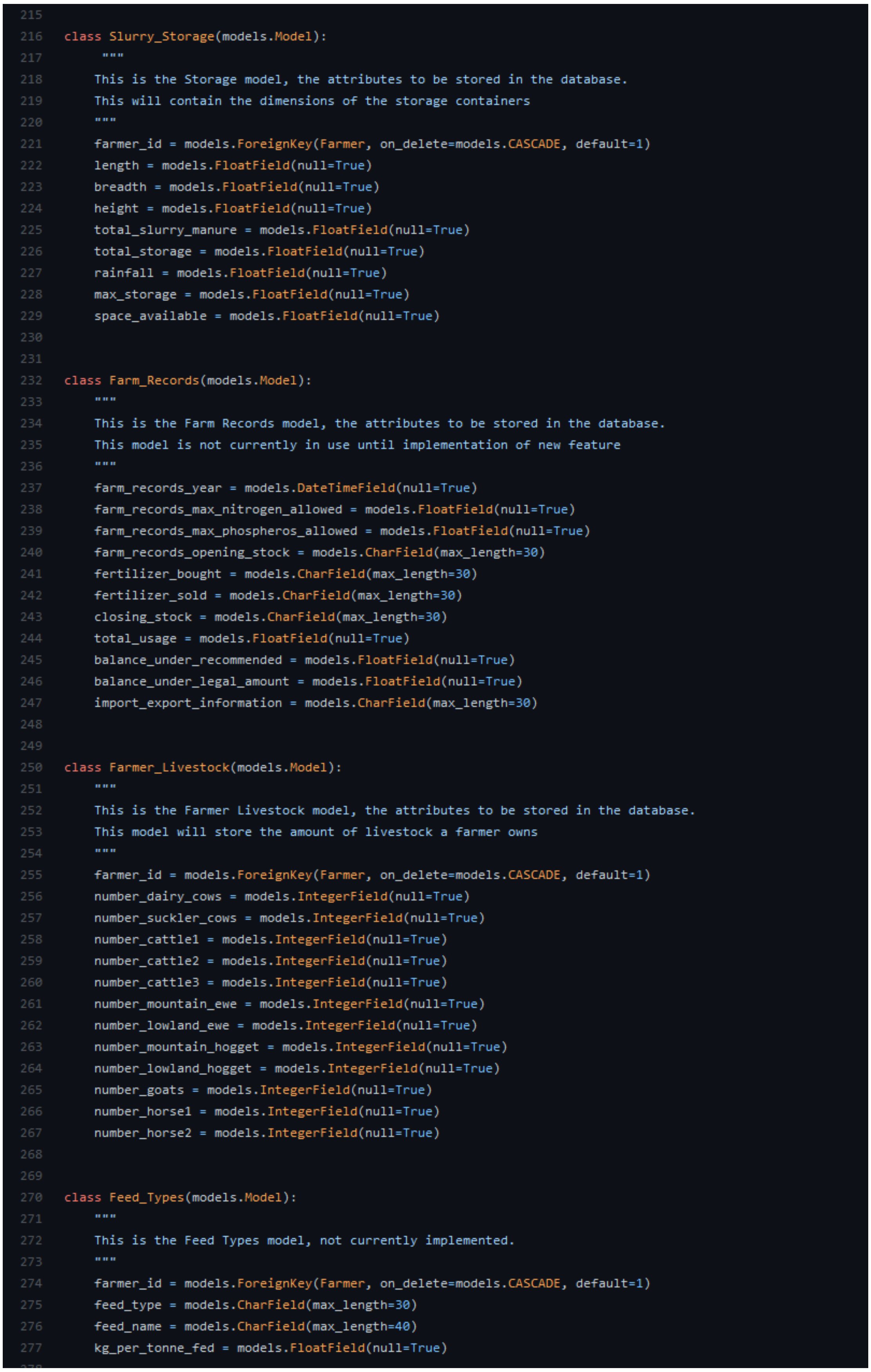

```python

class Farmer(models.Model):
    """
    This is the farmer model, the attributes to be stored in the database
    """
    name = models.CharField(max_length=30)
    address = models.CharField(max_length=30)
    county = models.CharField(max_length=30, choices=counties, null=True)
    date = models.DateField(null=True)
    herd_no = models.CharField(max_length=30, null=True, unique=True)
    is_assessed = models.IntegerField(null=True, default=0)
    zone = models.CharField(max_length=30)


class Grassland(models.Model):
    """
    This is the Grassland model, the attributes to be stored in the database.
    This will be changing after implementation of a new feature
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    owned_land = models.FloatField(null=True)
    rented_land = models.FloatField(null=True)
    time_rented = models.IntegerField(null=True)
    total_grass_area = models.FloatField(null=True)
    total_tillage_area = models.FloatField(null=True)
    total_land_area = models.FloatField(null=True)
    area_reseeded = models.FloatField(null=True)
    organicN = models.FloatField(null=True)
    organicP = models.FloatField(null=True)
    type_of_stock = models.CharField(max_length=30, null=True)
    type_of_feed = models.CharField(max_length=30, null=True)
    feed_name = models.CharField(max_length=30, null=True)
    feed_tonnage = models.CharField(max_length=30, null=True)
    number_of_animals = models.CharField(max_length=30, null=True)
    grassland_stocking_rate = models.FloatField(null=True)
    wholefarm_stocking_rate = models.FloatField(null=True)
    imports = models.FloatField(null=True)
    exports = models.FloatField(null=True)
    legalN_limit = models.FloatField(null=True)
    legalP_limit = models.FloatField(null=True)
    lsu = models.FloatField(null=True)

    # concentrateFed = models.FloatField(null=True)
    # soil_samples = models.CharField(max_length=30)
    # reseeding = models.CharField(max_length=30)
    # lime_required = models.FloatField(null=True)
    # enterprise = models.CharField(max_length=30)
    # sample_code = models.CharField(max_length=30, null=True)
    # date_taken = models.DateField(null=True)
    # expiry_date = models.DateField(null=True)
    # sample_area = models.FloatField(null=True)
    # ph = models.FloatField(null=True)
    # lime_required = models.FloatField(null=True)
    # p_value = models.FloatField(null=True)
    # p_index = models.FloatField(null=True)
    # k_value = models.FloatField(null=True)
    # k_index = models.FloatField(null=True)

```

Figure 15: Models.py



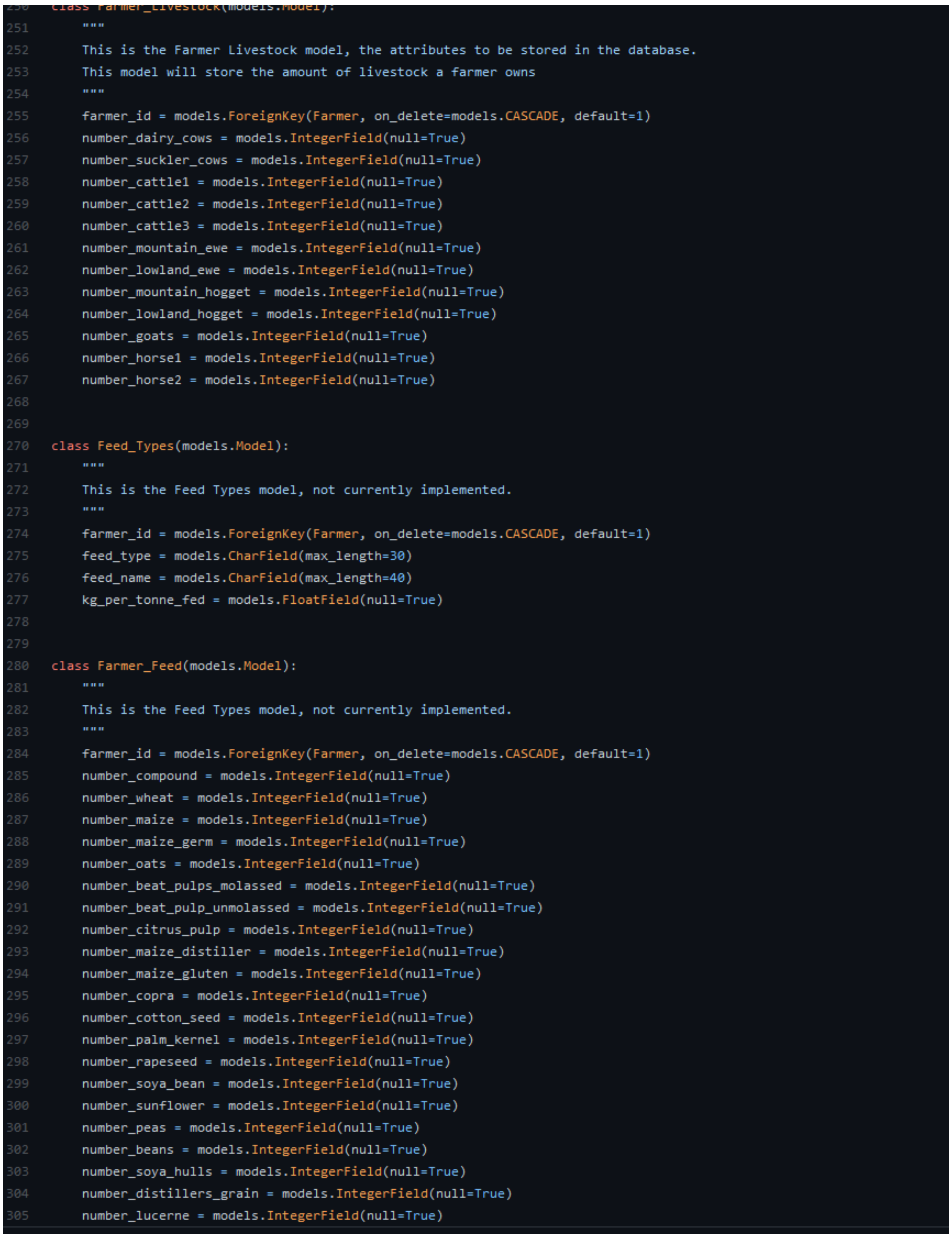
```
    # k_index = models.FloatField(null=True)

class Importation(models.Model):
    """
    This is the Importation model, the attributes to be stored in the database.
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    farmyard_manure = models.IntegerField(null=True)
    slurry = models.IntegerField(null=True)
    nitrates = models.IntegerField(null=True)
    phosphate = models.IntegerField(null=True)


class Exportation(models.Model):
    """
    This is the Exportation model, the attributes to be stored in the database.
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    farmyard_manure = models.IntegerField(null=True)
    slurry = models.IntegerField(null=True)
    nitrates = models.IntegerField(null=True)
    phosphate = models.IntegerField(null=True)


class Monthly_Livestock_Numbers(models.Model):
    """
    This is the Monthly livestock numbers model, the attributes to be stored
    in the database. This model also contains important information relating to
    each breed of livestock
    """
    monthly_livestock_numbers = models.TextField(null=True)
    type_of_animal = models.CharField(max_length=30)
    organic_nitrates = models.FloatField(null=True)
    organic_potassium = models.FloatField(null=True)
    lsu = models.FloatField(null=True)
    slurry_m3 = models.FloatField(null=True)
    manure_m3 = models.FloatField(null=True)


# class Tillage(models.Model):
    """
    This is the Feed Types model, not currently implemented.
    """
#     tillage_year = models.DateTimeField(null=True)
#     tillage_imports = models.FloatField(null=True)
#     area_tillage = models.FloatField(null=True)
#     area_grassland = models.FloatField(null=True)
#     organic_nitrates_applied = models.FloatField(null=True)
#     organic_phospherus_applied = models.FloatField(null=True)
#     applied_potassium = models.FloatField(null=True)
#     field = models.CharField(max_length=30)
#     fertilizer_allowed = models.FloatField(null=True)


# class Fertilzer_Plan(models.Model):
        """
        This is the Feed Types model, not currently implemented.
        """
#     opening_phospheros = models.FloatField(null=True)
#     opening_nitrogen = models.FloatField(null=True)
#     opening_stock = models.CharField(max_length=30)
#     planned_purchases = models.CharField(max_length=30)
#     lime = models.FloatField(null=True)
```

Figure 16: Models.py



```python

class Slurry_Storage(models.Model):
     """
    This is the Storage model, the attributes to be stored in the database.
    This will contain the dimensions of the storage containers
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    length = models.FloatField(null=True)
    breadth = models.FloatField(null=True)
    height = models.FloatField(null=True)
    total_slurry_manure = models.FloatField(null=True)
    total_storage = models.FloatField(null=True)
    rainfall = models.FloatField(null=True)
    max_storage = models.FloatField(null=True)
    space_available = models.FloatField(null=True)


class Farm_Records(models.Model):
    """
    This is the Farm Records model, the attributes to be stored in the database.
    This model is not currently in use until implementation of new feature
    """
    farm_records_year = models.DateTimeField(null=True)
    farm_records_max_nitrogen_allowed = models.FloatField(null=True)
    farm_records_max_phospheros_allowed = models.FloatField(null=True)
    farm_records_opening_stock = models.CharField(max_length=30)
    fertilizer_bought = models.CharField(max_length=30)
    fertilizer_sold = models.CharField(max_length=30)
    closing_stock = models.CharField(max_length=30)
    total_usage = models.FloatField(null=True)
    balance_under_recommended = models.FloatField(null=True)
    balance_under_legal_amount = models.FloatField(null=True)
    import_export_information = models.CharField(max_length=30)


class Farmer_Livestock(models.Model):
    """
    This is the Farmer Livestock model, the attributes to be stored in the database.
    This model will store the amount of livestock a farmer owns
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    number_dairy_cows = models.IntegerField(null=True)
    number_suckler_cows = models.IntegerField(null=True)
    number_cattle1 = models.IntegerField(null=True)
    number_cattle2 = models.IntegerField(null=True)
    number_cattle3 = models.IntegerField(null=True)
    number_mountain_ewe = models.IntegerField(null=True)
    number_lowland_ewe = models.IntegerField(null=True)
    number_mountain_hogget = models.IntegerField(null=True)
    number_lowland_hogget = models.IntegerField(null=True)
    number_goats = models.IntegerField(null=True)
    number_horse1 = models.IntegerField(null=True)
    number_horse2 = models.IntegerField(null=True)


class Feed_Types(models.Model):
    """
    This is the Feed Types model, not currently implemented.
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    feed_type = models.CharField(max_length=30)
    feed_name = models.CharField(max_length=40)
    kg_per_tonne_fed = models.FloatField(null=True)
```

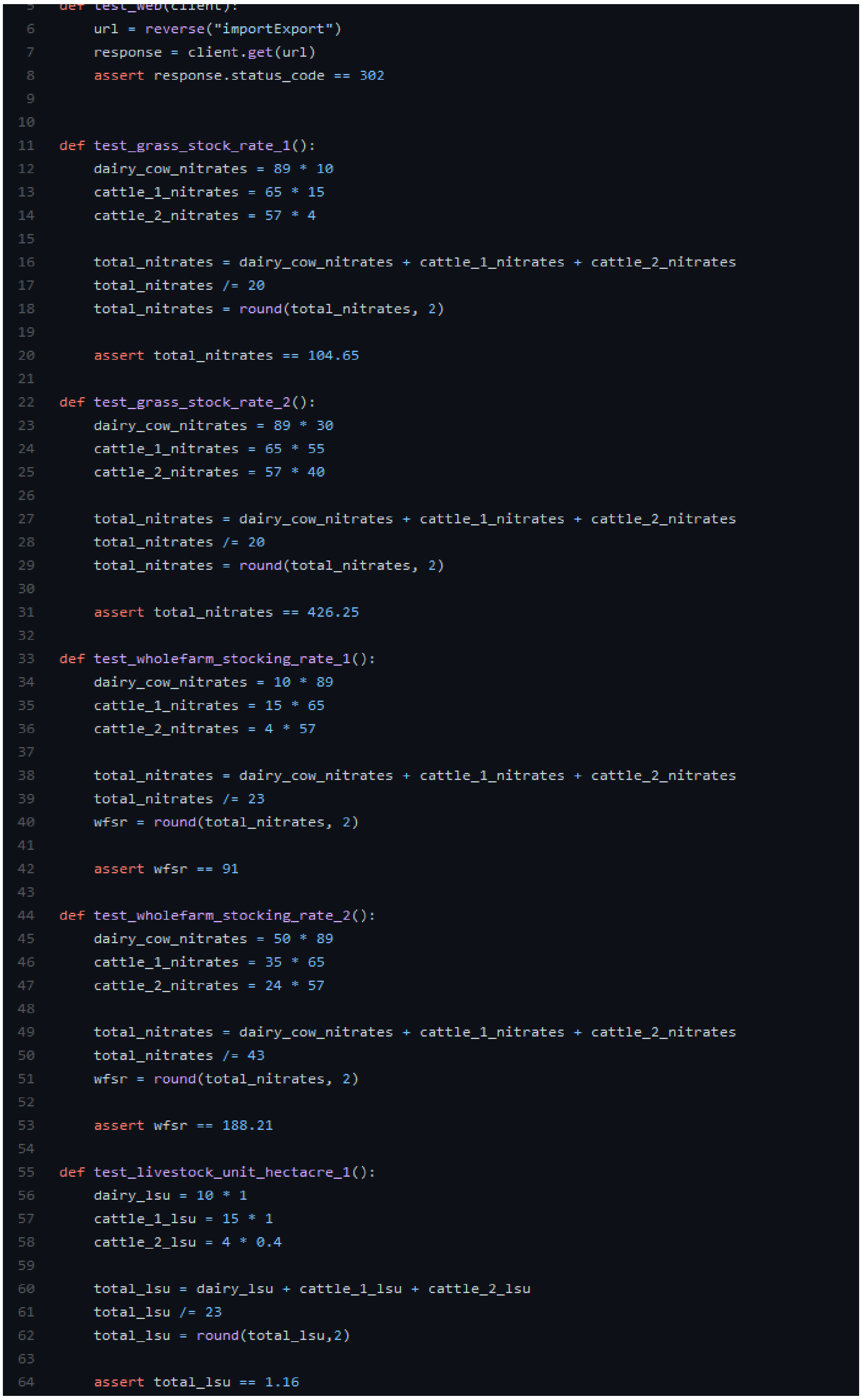
Figure 17: Models.py



```python
class Farmer_Livestock(models.Model):
    """
    This is the Farmer Livestock model, the attributes to be stored in the database.
    This model will store the amount of livestock a farmer owns
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    number_dairy_cows = models.IntegerField(null=True)
    number_suckler_cows = models.IntegerField(null=True)
    number_cattle1 = models.IntegerField(null=True)
    number_cattle2 = models.IntegerField(null=True)
    number_cattle3 = models.IntegerField(null=True)
    number_mountain_ewe = models.IntegerField(null=True)
    number_lowland_ewe = models.IntegerField(null=True)
    number_mountain_hogget = models.IntegerField(null=True)
    number_lowland_hogget = models.IntegerField(null=True)
    number_goats = models.IntegerField(null=True)
    number_horse1 = models.IntegerField(null=True)
    number_horse2 = models.IntegerField(null=True)


class Feed_Types(models.Model):
    """
    This is the Feed Types model, not currently implemented.
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    feed_type = models.CharField(max_length=30)
    feed_name = models.CharField(max_length=40)
    kg_per_tonne_fed = models.FloatField(null=True)


class Farmer_Feed(models.Model):
    """
    This is the Feed Types model, not currently implemented.
    """
    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
    number_compound = models.IntegerField(null=True)
    number_wheat = models.IntegerField(null=True)
    number_maize = models.IntegerField(null=True)
    number_maize_germ = models.IntegerField(null=True)
    number_oats = models.IntegerField(null=True)
    number_beat_pulps_molassed = models.IntegerField(null=True)
    number_beat_pulp_unmolassed = models.IntegerField(null=True)
    number_citrus_pulp = models.IntegerField(null=True)
    number_maize_distiller = models.IntegerField(null=True)
    number_maize_gluten = models.IntegerField(null=True)
    number_copra = models.IntegerField(null=True)
    number_cotton_seed = models.IntegerField(null=True)
    number_palm_kernel = models.IntegerField(null=True)
    number_rapeseed = models.IntegerField(null=True)
    number_soya_bean = models.IntegerField(null=True)
    number_sunflower = models.IntegerField(null=True)
    number_peas = models.IntegerField(null=True)
    number_beans = models.IntegerField(null=True)
    number_soya_hulls = models.IntegerField(null=True)
    number_distillers_grain = models.IntegerField(null=True)
    number_lucerne = models.IntegerField(null=True)
```

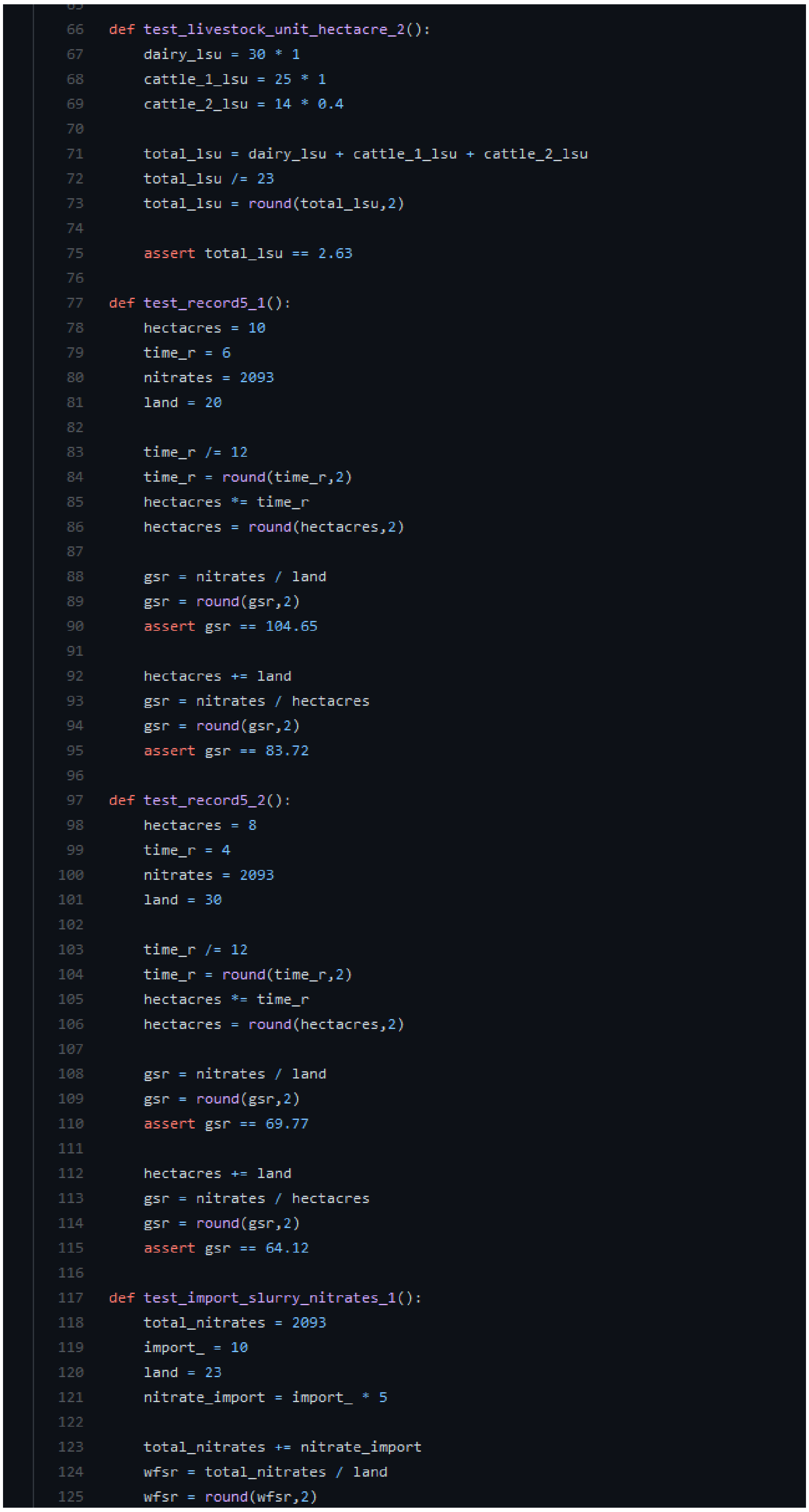
Figure 18: Models.py

Tests



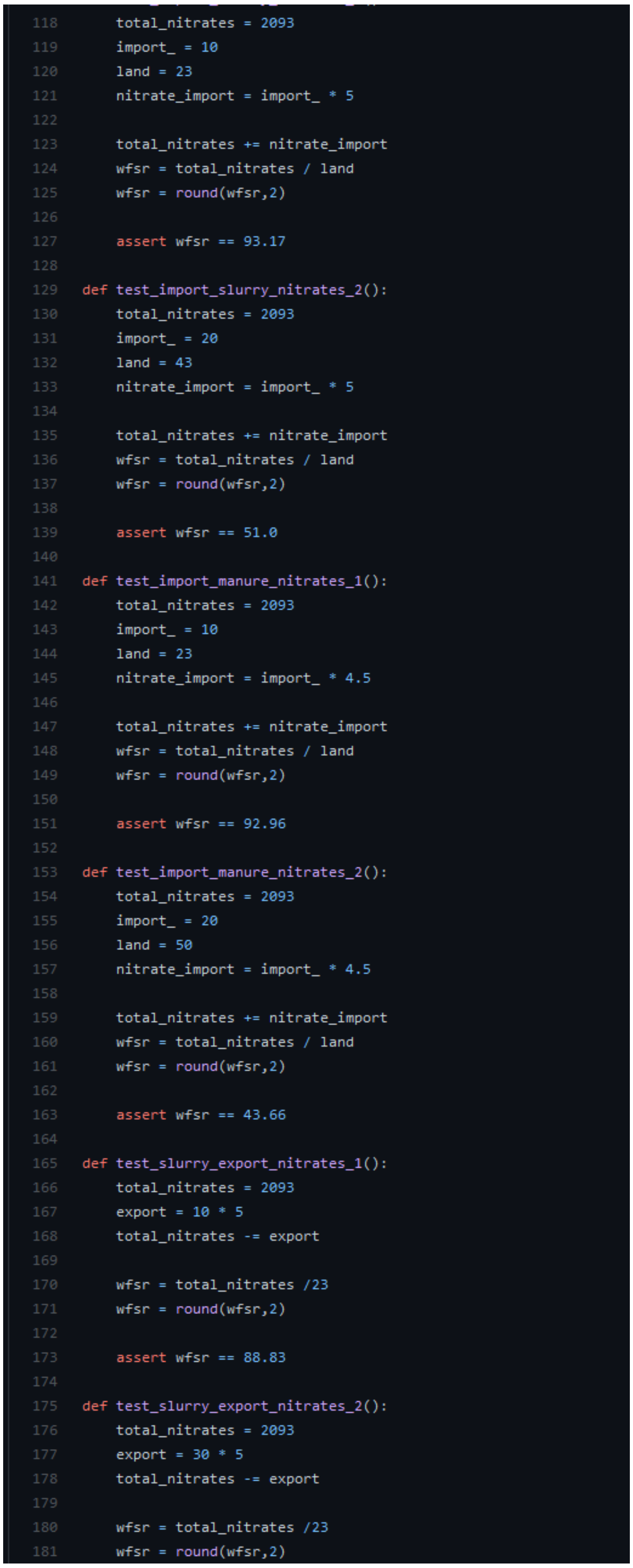
```python
def test_web(client):
    url = reverse("importExport")
    response = client.get(url)
    assert response.status_code == 302


def test_grass_stock_rate_1():
    dairy_cow_nitrates = 89 * 10
    cattle_1_nitrates = 65 * 15
    cattle_2_nitrates = 57 * 4

    total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
    total_nitrates /= 20
    total_nitrates = round(total_nitrates, 2)

    assert total_nitrates == 104.65

def test_grass_stock_rate_2():
    dairy_cow_nitrates = 89 * 30
    cattle_1_nitrates = 65 * 55
    cattle_2_nitrates = 57 * 40

    total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
    total_nitrates /= 20
    total_nitrates = round(total_nitrates, 2)

    assert total_nitrates == 426.25

def test_wholefarm_stocking_rate_1():
    dairy_cow_nitrates = 10 * 89
    cattle_1_nitrates = 15 * 65
    cattle_2_nitrates = 4 * 57

    total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
    total_nitrates /= 23
    wfsr = round(total_nitrates, 2)

    assert wfsr == 91

def test_wholefarm_stocking_rate_2():
    dairy_cow_nitrates = 50 * 89
    cattle_1_nitrates = 35 * 65
    cattle_2_nitrates = 24 * 57

    total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
    total_nitrates /= 43
    wfsr = round(total_nitrates, 2)

    assert wfsr == 188.21

def test_livestock_unit_hectacre_1():
    dairy_lsu = 10 * 1
    cattle_1_lsu = 15 * 1
    cattle_2_lsu = 4 * 0.4

    total_lsu = dairy_lsu + cattle_1_lsu + cattle_2_lsu
    total_lsu /= 23
    total_lsu = round(total_lsu,2)

    assert total_lsu == 1.16
```

Figure 19: Test.py



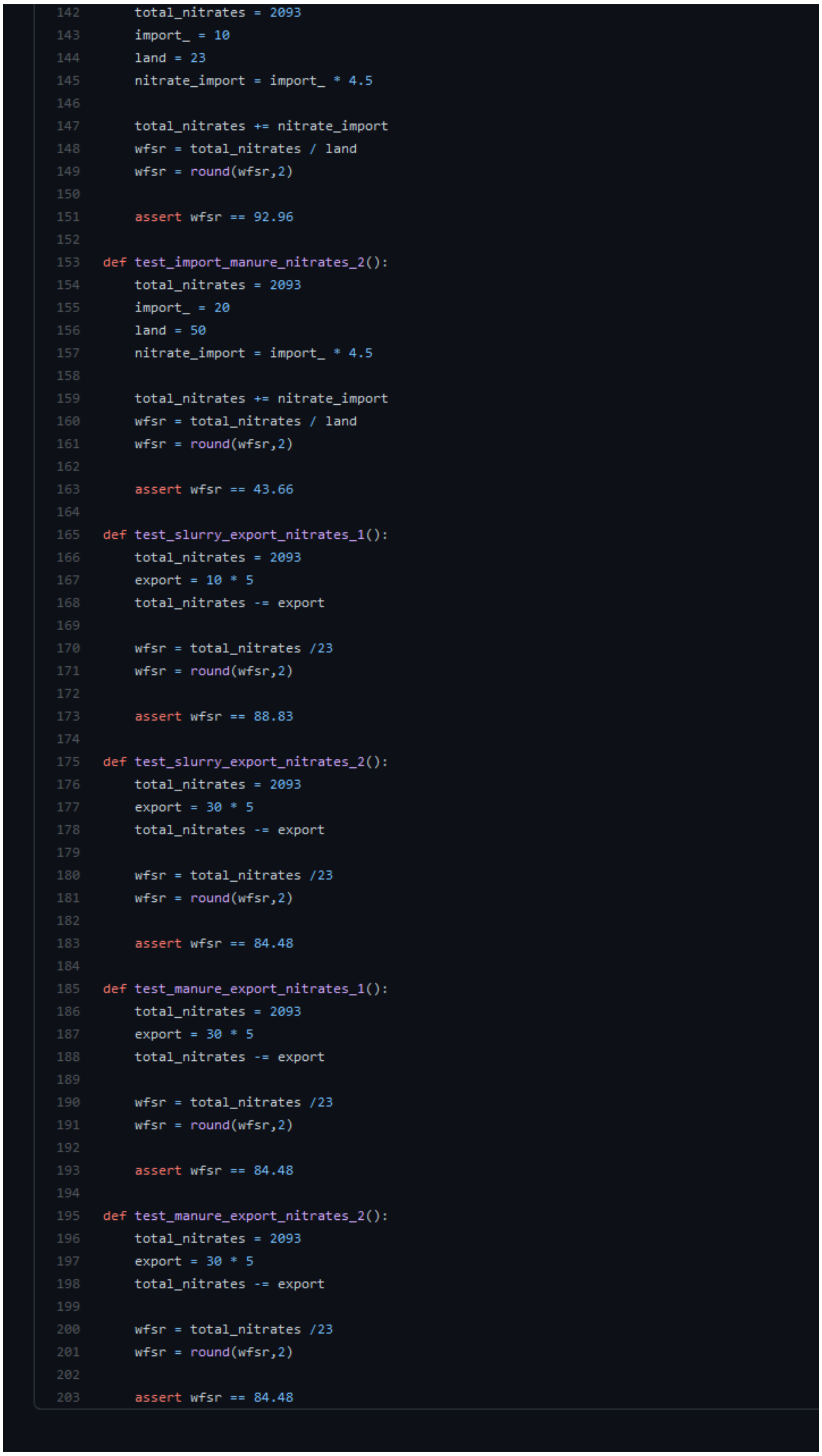
```python
def test_livestock_unit_hectacre_2():
    dairy_lsu = 30 * 1
    cattle_1_lsu = 25 * 1
    cattle_2_lsu = 14 * 0.4

    total_lsu = dairy_lsu + cattle_1_lsu + cattle_2_lsu
    total_lsu /= 23
    total_lsu = round(total_lsu,2)

    assert total_lsu == 2.63

def test_record5_1():
    hectacres = 10
    time_r = 6
    nitrates = 2093
    land = 20

    time_r /= 12
    time_r = round(time_r,2)
    hectacres *= time_r
    hectacres = round(hectacres,2)

    gsr = nitrates / land
    gsr = round(gsr,2)
    assert gsr == 104.65

    hectacres += land
    gsr = nitrates / hectacres
    gsr = round(gsr,2)
    assert gsr == 83.72

def test_record5_2():
    hectacres = 8
    time_r = 4
    nitrates = 2093
    land = 30

    time_r /= 12
    time_r = round(time_r,2)
    hectacres *= time_r
    hectacres = round(hectacres,2)

    gsr = nitrates / land
    gsr = round(gsr,2)
    assert gsr == 69.77

    hectacres += land
    gsr = nitrates / hectacres
    gsr = round(gsr,2)
    assert gsr == 64.12

def test_import_slurry_nitrates_1():
    total_nitrates = 2093
    import_ = 10
    land = 23
    nitrate_import = import_ * 5

    total_nitrates += nitrate_import
    wfsr = total_nitrates / land
    wfsr = round(wfsr,2)
```

Figure 20: Test.py



```python
    total_nitrates = 2093
    import_ = 10
    land = 23
    nitrate_import = import_ * 5

    total_nitrates += nitrate_import
    wfsr = total_nitrates / land
    wfsr = round(wfsr,2)

    assert wfsr == 93.17

def test_import_slurry_nitrates_2():
    total_nitrates = 2093
    import_ = 20
    land = 43
    nitrate_import = import_ * 5

    total_nitrates += nitrate_import
    wfsr = total_nitrates / land
    wfsr = round(wfsr,2)

    assert wfsr == 51.0

def test_import_manure_nitrates_1():
    total_nitrates = 2093
    import_ = 10
    land = 23
    nitrate_import = import_ * 4.5

    total_nitrates += nitrate_import
    wfsr = total_nitrates / land
    wfsr = round(wfsr,2)

    assert wfsr == 92.96

def test_import_manure_nitrates_2():
    total_nitrates = 2093
    import_ = 20
    land = 50
    nitrate_import = import_ * 4.5

    total_nitrates += nitrate_import
    wfsr = total_nitrates / land
    wfsr = round(wfsr,2)

    assert wfsr == 43.66

def test_slurry_export_nitrates_1():
    total_nitrates = 2093
    export = 10 * 5
    total_nitrates -= export

    wfsr = total_nitrates /23
    wfsr = round(wfsr,2)

    assert wfsr == 88.83

def test_slurry_export_nitrates_2():
    total_nitrates = 2093
    export = 30 * 5
    total_nitrates -= export

    wfsr = total_nitrates /23
    wfsr = round(wfsr,2)
```

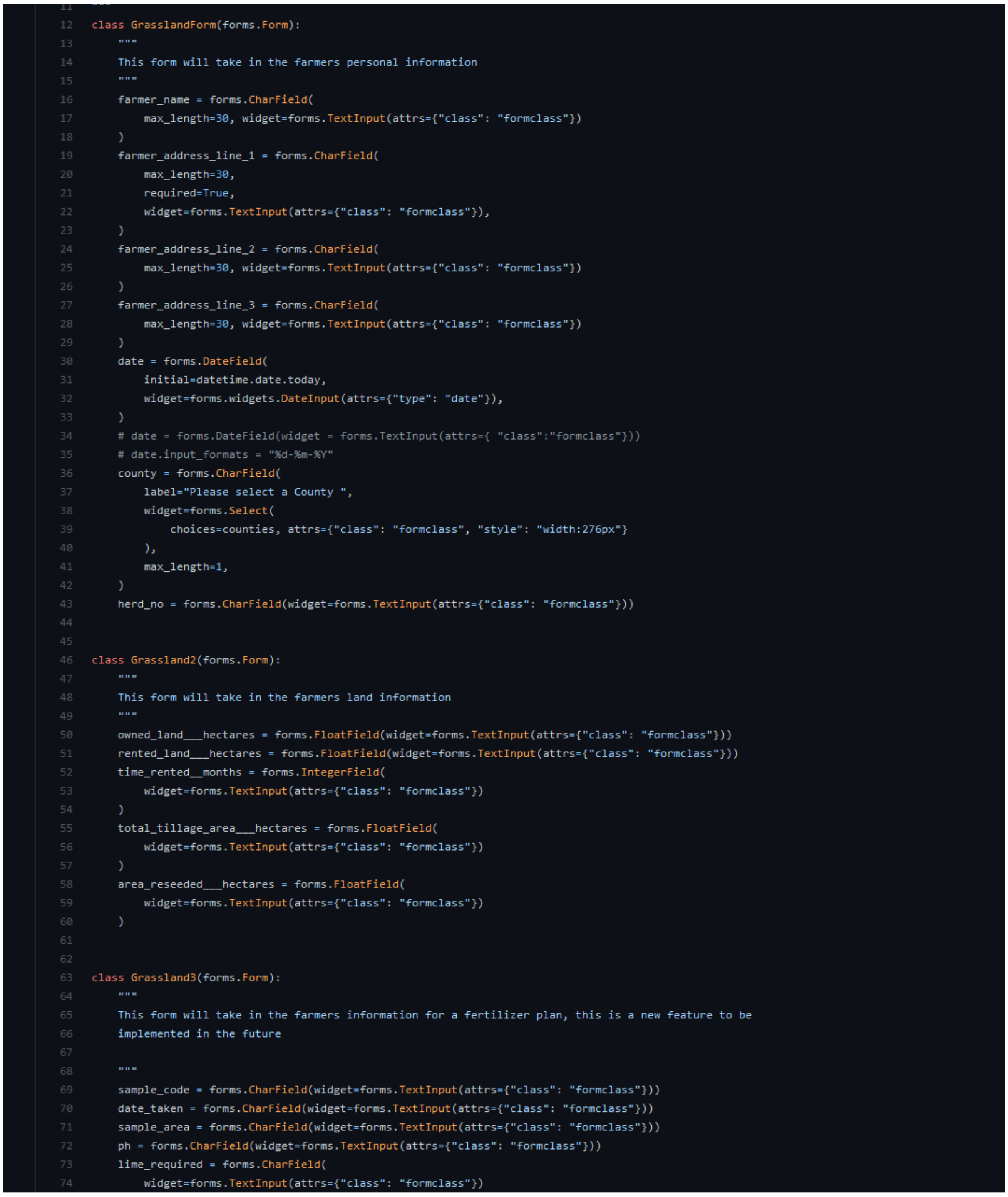
Figure 21: Test.py



```python
    total_nitrates = 2093
    import_ = 10
    land = 23
    nitrate_import = import_ * 4.5

    total_nitrates += nitrate_import
    wfsr = total_nitrates / land
    wfsr = round(wfsr,2)

    assert wfsr == 92.96

def test_import_manure_nitrates_2():
    total_nitrates = 2093
    import_ = 20
    land = 50
    nitrate_import = import_ * 4.5

    total_nitrates += nitrate_import
    wfsr = total_nitrates / land
    wfsr = round(wfsr,2)

    assert wfsr == 43.66

def test_slurry_export_nitrates_1():
    total_nitrates = 2093
    export = 10 * 5
    total_nitrates -= export

    wfsr = total_nitrates /23
    wfsr = round(wfsr,2)

    assert wfsr == 88.83

def test_slurry_export_nitrates_2():
    total_nitrates = 2093
    export = 30 * 5
    total_nitrates -= export

    wfsr = total_nitrates /23
    wfsr = round(wfsr,2)

    assert wfsr == 84.48

def test_manure_export_nitrates_1():
    total_nitrates = 2093
    export = 30 * 5
    total_nitrates -= export

    wfsr = total_nitrates /23
    wfsr = round(wfsr,2)

    assert wfsr == 84.48

def test_manure_export_nitrates_2():
    total_nitrates = 2093
    export = 30 * 5
    total_nitrates -= export

    wfsr = total_nitrates /23
    wfsr = round(wfsr,2)

    assert wfsr == 84.48
```

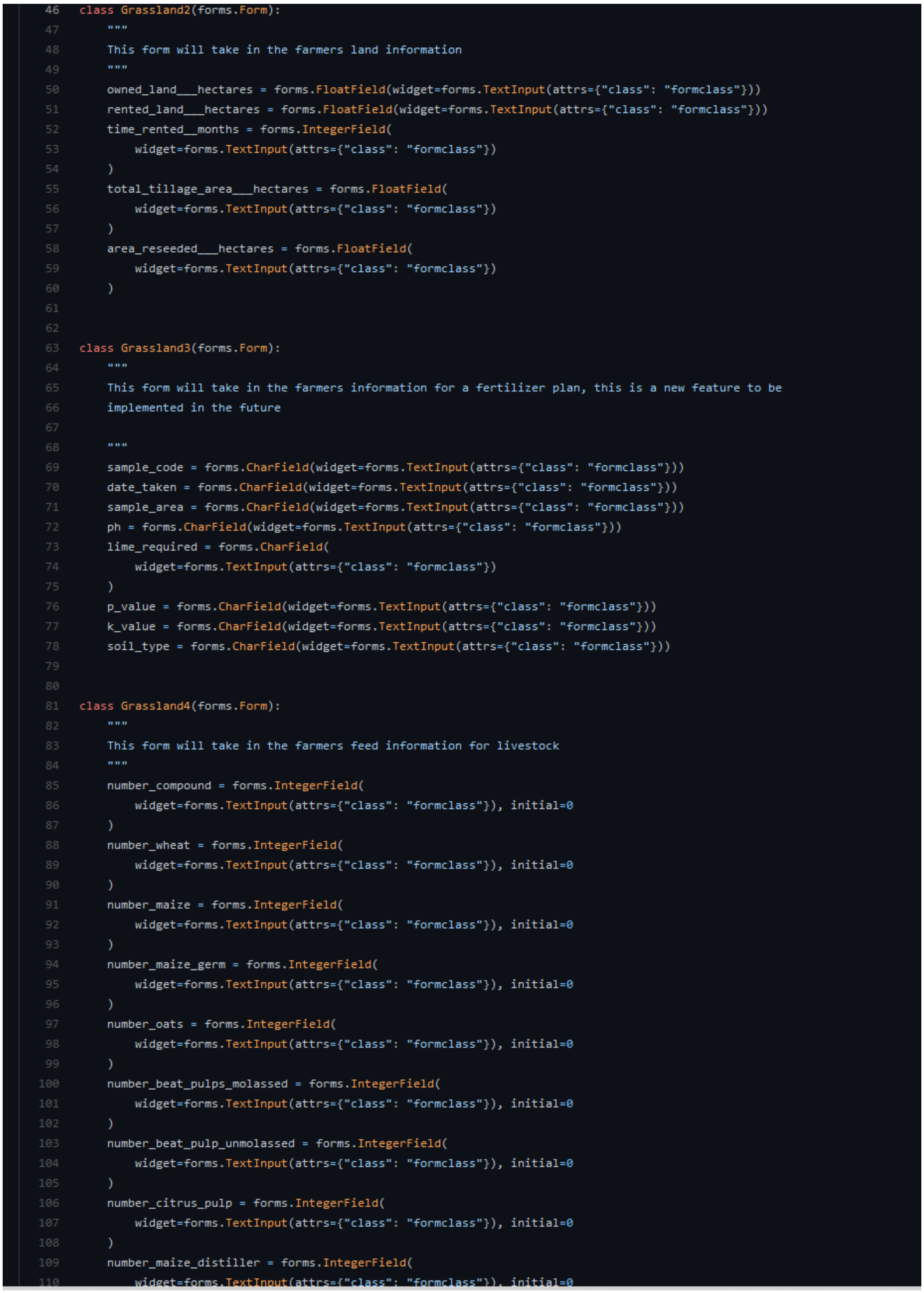
Figure 22:Test.py

Forms.py



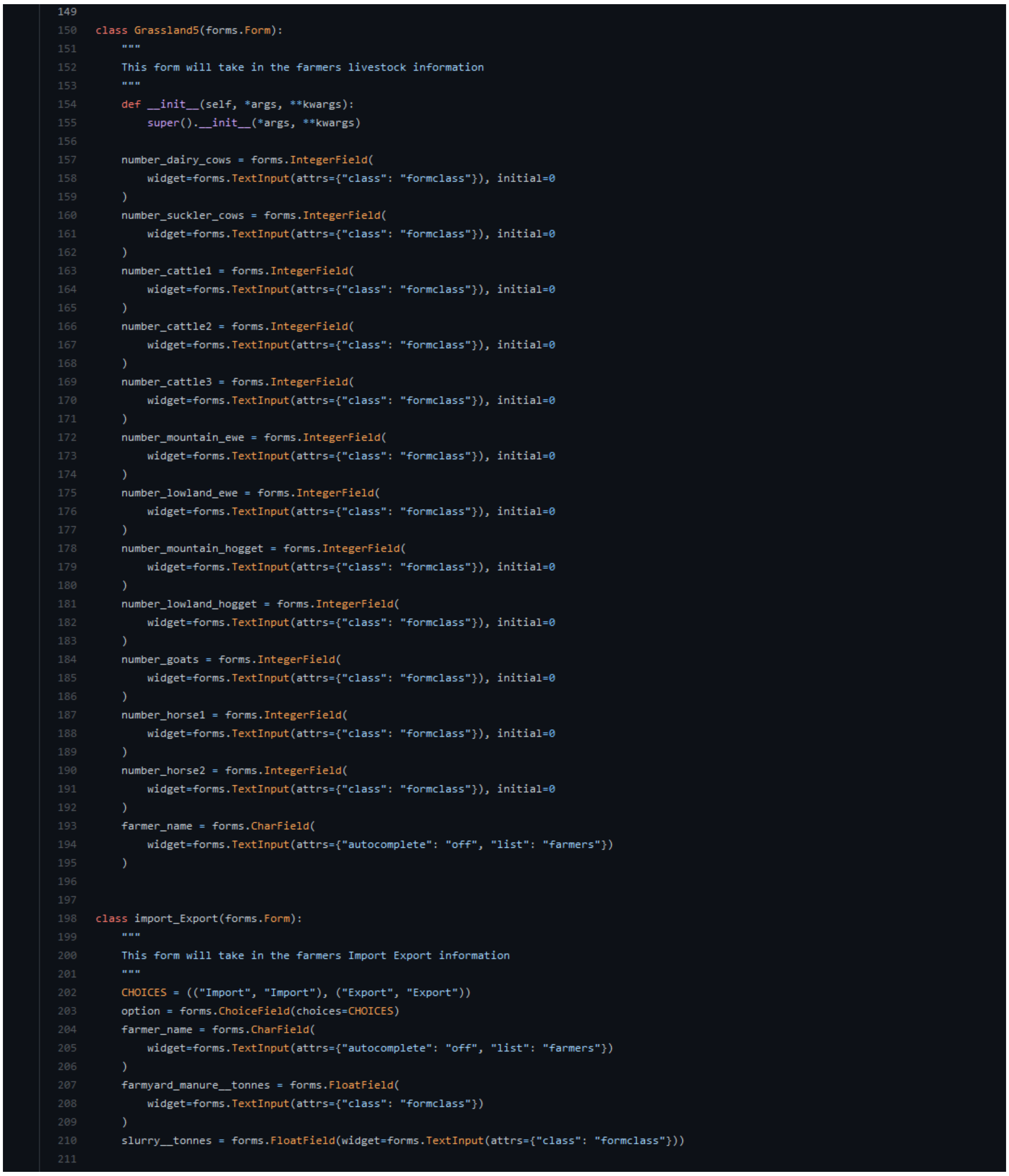
```python
class GrasslandForm(forms.Form):
    """
    This form will take in the farmers personal information
    """
    farmer_name = forms.CharField(
        max_length=30, widget=forms.TextInput(attrs={"class": "formclass"})
    )
    farmer_address_line_1 = forms.CharField(
        max_length=30,
        required=True,
        widget=forms.TextInput(attrs={"class": "formclass"}),
    )
    farmer_address_line_2 = forms.CharField(
        max_length=30, widget=forms.TextInput(attrs={"class": "formclass"})
    )
    farmer_address_line_3 = forms.CharField(
        max_length=30, widget=forms.TextInput(attrs={"class": "formclass"})
    )
    date = forms.DateField(
        initial=datetime.date.today,
        widget=forms.widgets.DateInput(attrs={"type": "date"}),
    )
    # date = forms.DateField(widget = forms.TextInput(attrs={ "class":"formclass"}))
    # date.input_formats = "%d-%m-%Y"
    county = forms.CharField(
        label="Please select a County ",
        widget=forms.Select(
            choices=counties, attrs={"class": "formclass", "style": "width:276px"}
        ),
        max_length=1,
    )
    herd_no = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))


class Grassland2(forms.Form):
    """
    This form will take in the farmers land information
    """
    owned_land__hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
    rented_land__hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
    time_rented__months = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )
    total_tillage_area__hectares = forms.FloatField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )
    area_reseeded__hectares = forms.FloatField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )


class Grassland3(forms.Form):
    """
    This form will take in the farmers information for a fertilizer plan, this is a new feature to be
    implemented in the future

    """
    sample_code = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    date_taken = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    sample_area = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    ph = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    lime_required = forms.CharField(
        widget=forms.TextInput(attrs={"class": "formclass"})
```

Figure 23: Forms.py



```python
class Grassland2(forms.Form):
    """
    This form will take in the farmers land information
    """
    owned_land___hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
    rented_land___hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
    time_rented__months = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )
    total_tillage_area___hectares = forms.FloatField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )
    area_reseeded___hectares = forms.FloatField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )


class Grassland3(forms.Form):
    """
    This form will take in the farmers information for a fertilizer plan, this is a new feature to be
    implemented in the future

    """
    sample_code = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    date_taken = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    sample_area = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    ph = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    lime_required = forms.CharField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )
    p_value = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    k_value = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
    soil_type = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))


class Grassland4(forms.Form):
    """
    This form will take in the farmers feed information for livestock
    """
    number_compound = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_wheat = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_maize = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_maize_germ = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_oats = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_beat_pulps_molassed = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_beat_pulp_unmolassed = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_citrus_pulp = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_maize_distiller = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
```

Figure 24: Forms. py



```python

class Grassland5(forms.Form):
    """
    This form will take in the farmers livestock information
    """
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    number_dairy_cows = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_suckler_cows = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_cattle1 = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_cattle2 = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_cattle3 = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_mountain_ewe = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_lowland_ewe = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_mountain_hogget = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_lowland_hogget = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_goats = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_horse1 = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    number_horse2 = forms.IntegerField(
        widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
    )
    farmer_name = forms.CharField(
        widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
    )


class import_Export(forms.Form):
    """
    This form will take in the farmers Import Export information
    """
    CHOICES = (("Import", "Import"), ("Export", "Export"))
    option = forms.ChoiceField(choices=CHOICES)
    farmer_name = forms.CharField(
        widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
    )
    farmyard_manure__tonnes = forms.FloatField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )
    slurry__tonnes = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))

```

Figure 25: Forms.py

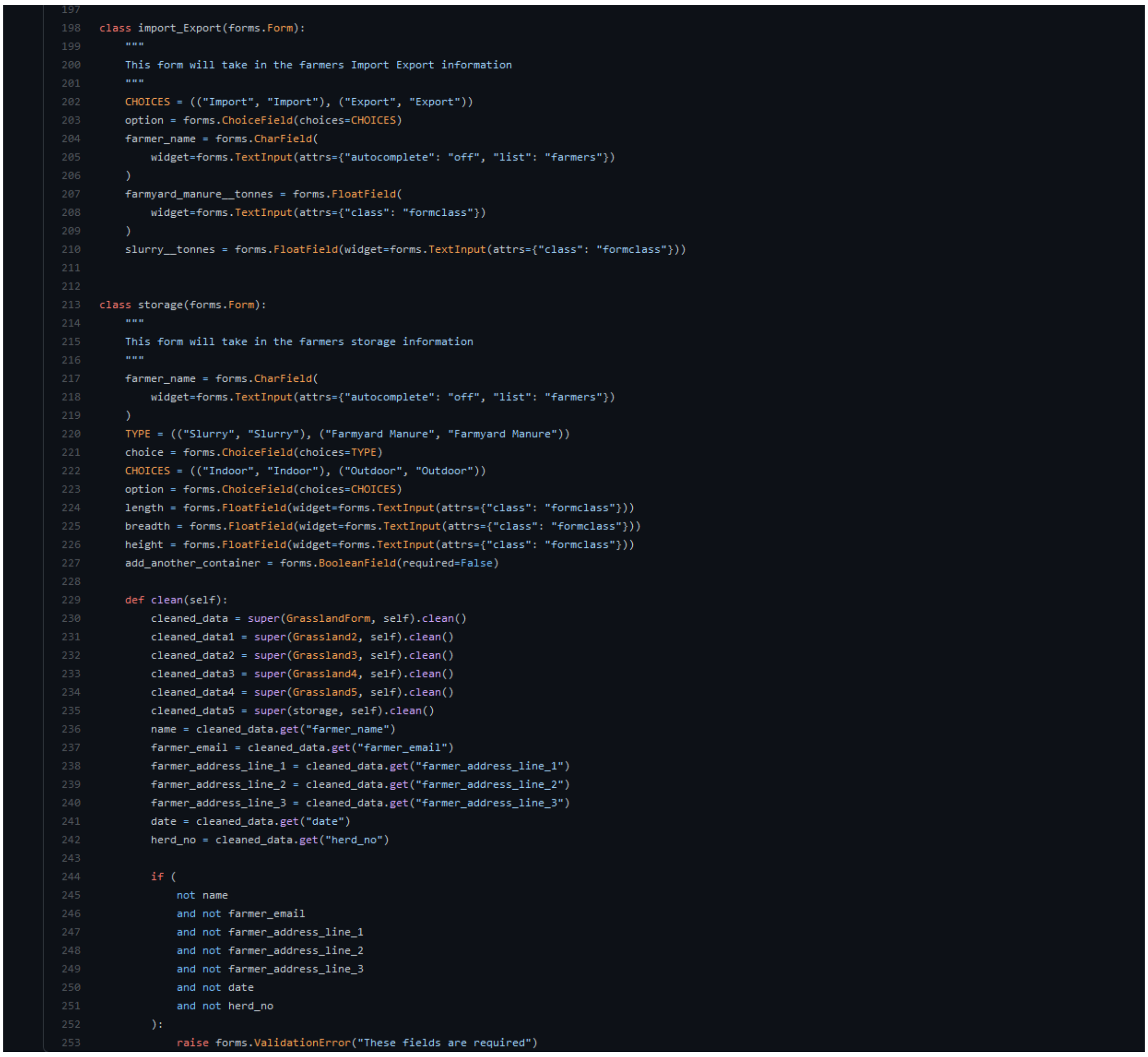
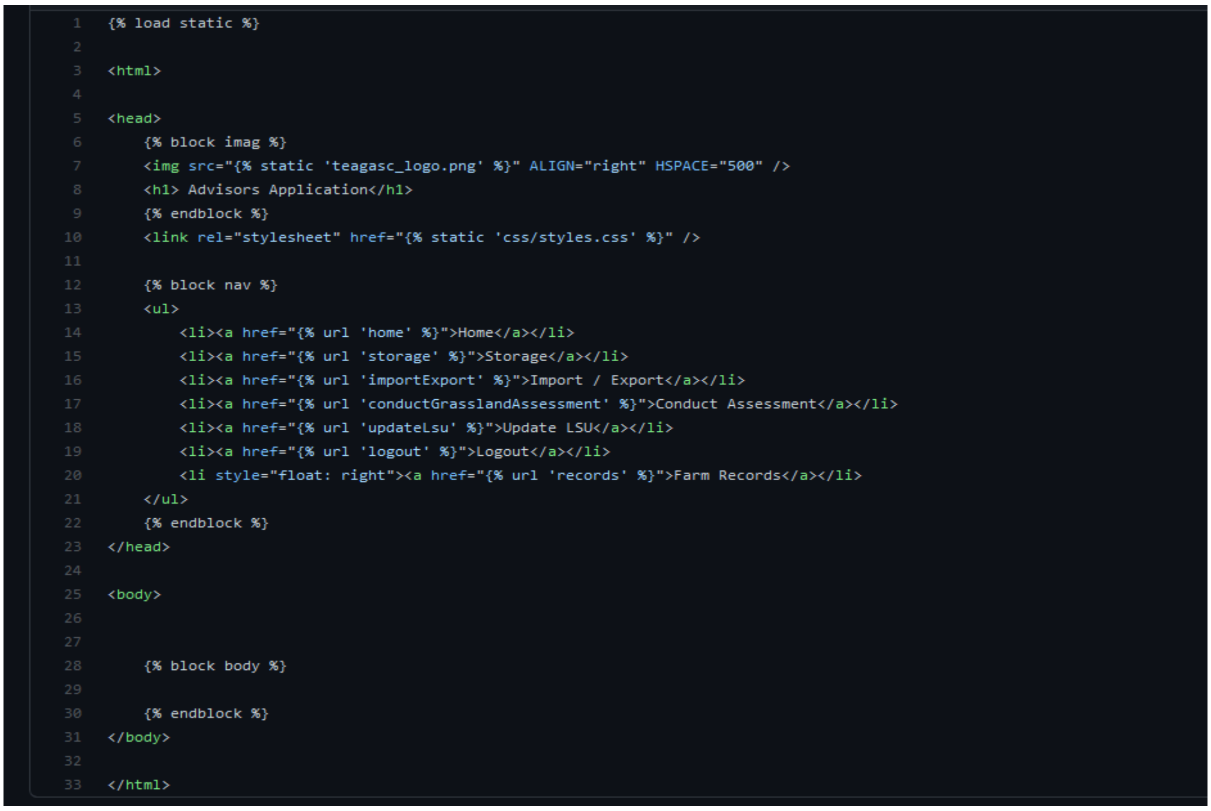

```python
class import_Export(forms.Form):
    """
    This form will take in the farmers Import Export information
    """
    CHOICES = (("Import", "Import"), ("Export", "Export"))
    option = forms.ChoiceField(choices=CHOICES)
    farmer_name = forms.CharField(
        widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
    )
    farmyard_manure__tonnes = forms.FloatField(
        widget=forms.TextInput(attrs={"class": "formclass"})
    )
    slurry__tonnes = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))


class storage(forms.Form):
    """
    This form will take in the farmers storage information
    """
    farmer_name = forms.CharField(
        widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
    )
    TYPE = (("Slurry", "Slurry"), ("Farmyard Manure", "Farmyard Manure"))
    choice = forms.ChoiceField(choices=TYPE)
    CHOICES = (("Indoor", "Indoor"), ("Outdoor", "Outdoor"))
    option = forms.ChoiceField(choices=CHOICES)
    length = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
    breadth = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
    height = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
    add_another_container = forms.BooleanField(required=False)

    def clean(self):
        cleaned_data = super(GrasslandForm, self).clean()
        cleaned_data1 = super(Grassland2, self).clean()
        cleaned_data2 = super(Grassland3, self).clean()
        cleaned_data3 = super(Grassland4, self).clean()
        cleaned_data4 = super(Grassland5, self).clean()
        cleaned_data5 = super(storage, self).clean()
        name = cleaned_data.get("farmer_name")
        farmer_email = cleaned_data.get("farmer_email")
        farmer_address_line_1 = cleaned_data.get("farmer_address_line_1")
        farmer_address_line_2 = cleaned_data.get("farmer_address_line_2")
        farmer_address_line_3 = cleaned_data.get("farmer_address_line_3")
        date = cleaned_data.get("date")
        herd_no = cleaned_data.get("herd_no")

        if (
            not name
            and not farmer_email
            and not farmer_address_line_1
            and not farmer_address_line_2
            and not farmer_address_line_3
            and not date
            and not herd_no
        ):
            raise forms.ValidationError("These fields are required")
```

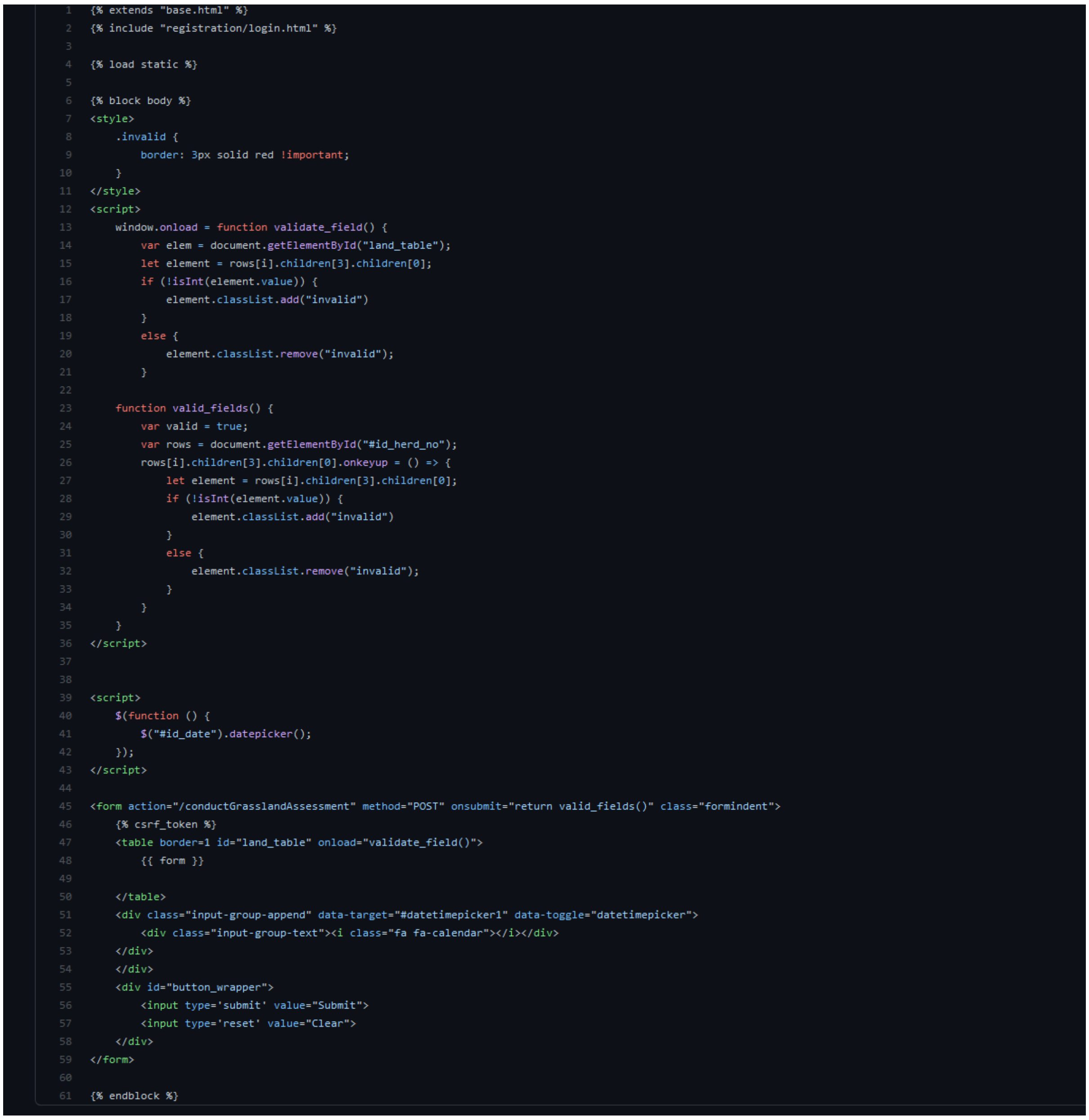
Figure 26: Forms.py

Templates



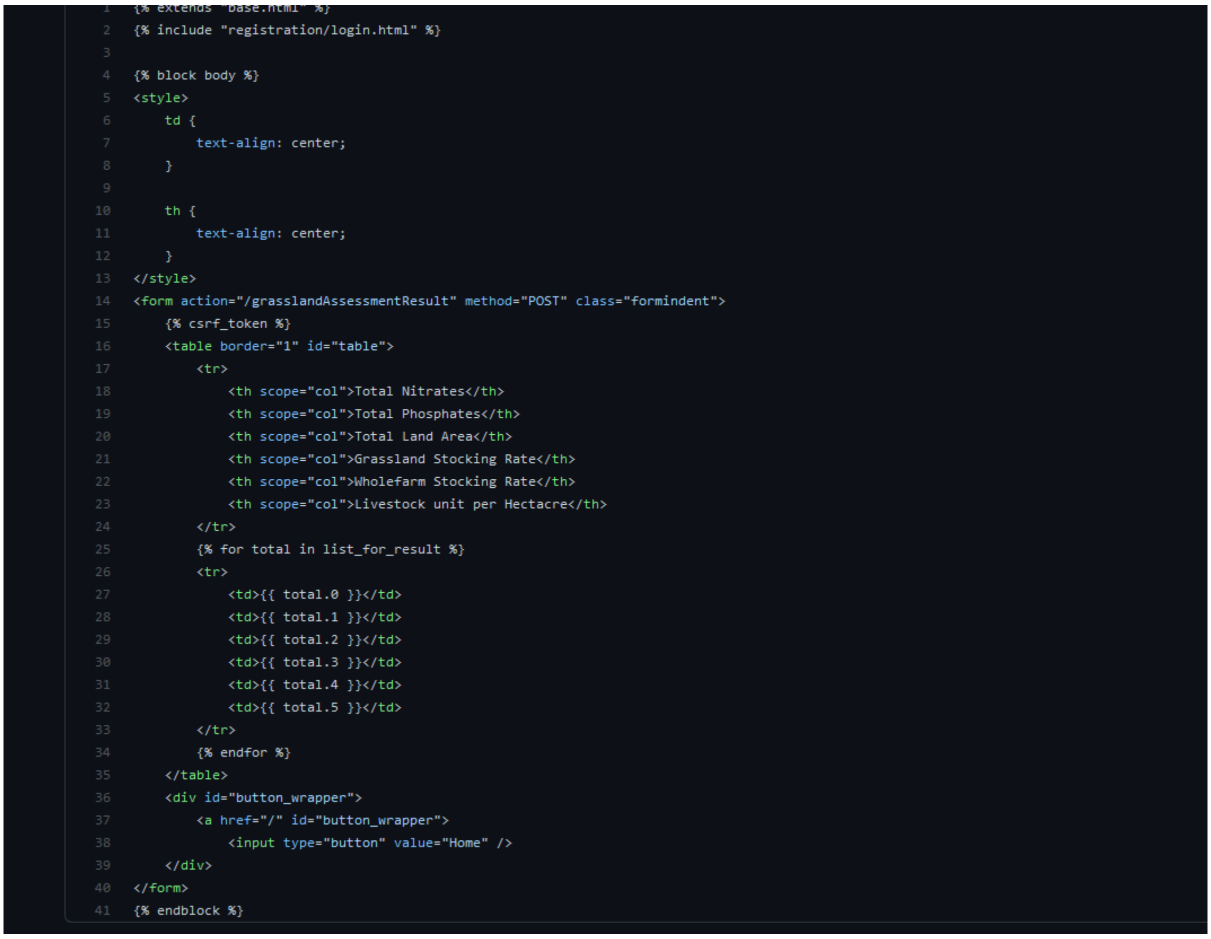
```
{% load static %}

<html>

<head>
    {% block imag %}
    <img src="{% static 'teagasc_logo.png' %}" ALIGN="right" HSPACE="500" />
    <h1> Advisors Application</h1>
    {% endblock %}
    <link rel="stylesheet" href="{% static 'css/styles.css' %}" />

    {% block nav %}
    <ul>
        <li><a href="{% url 'home' %}">Home</a></li>
        <li><a href="{% url 'storage' %}">Storage</a></li>
        <li><a href="{% url 'importExport' %}">Import / Export</a></li>
        <li><a href="{% url 'conductGrasslandAssessment' %}">Conduct Assessment</a></li>
        <li><a href="{% url 'updateLsu' %}">Update LSU</a></li>
        <li><a href="{% url 'logout' %}">Logout</a></li>
        <li style="float: right"><a href="{% url 'records' %}">Farm Records</a></li>
    </ul>
    {% endblock %}
</head>

<body>


    {% block body %}

    {% endblock %}
</body>

</html>
```

Figure 27: Templates.py



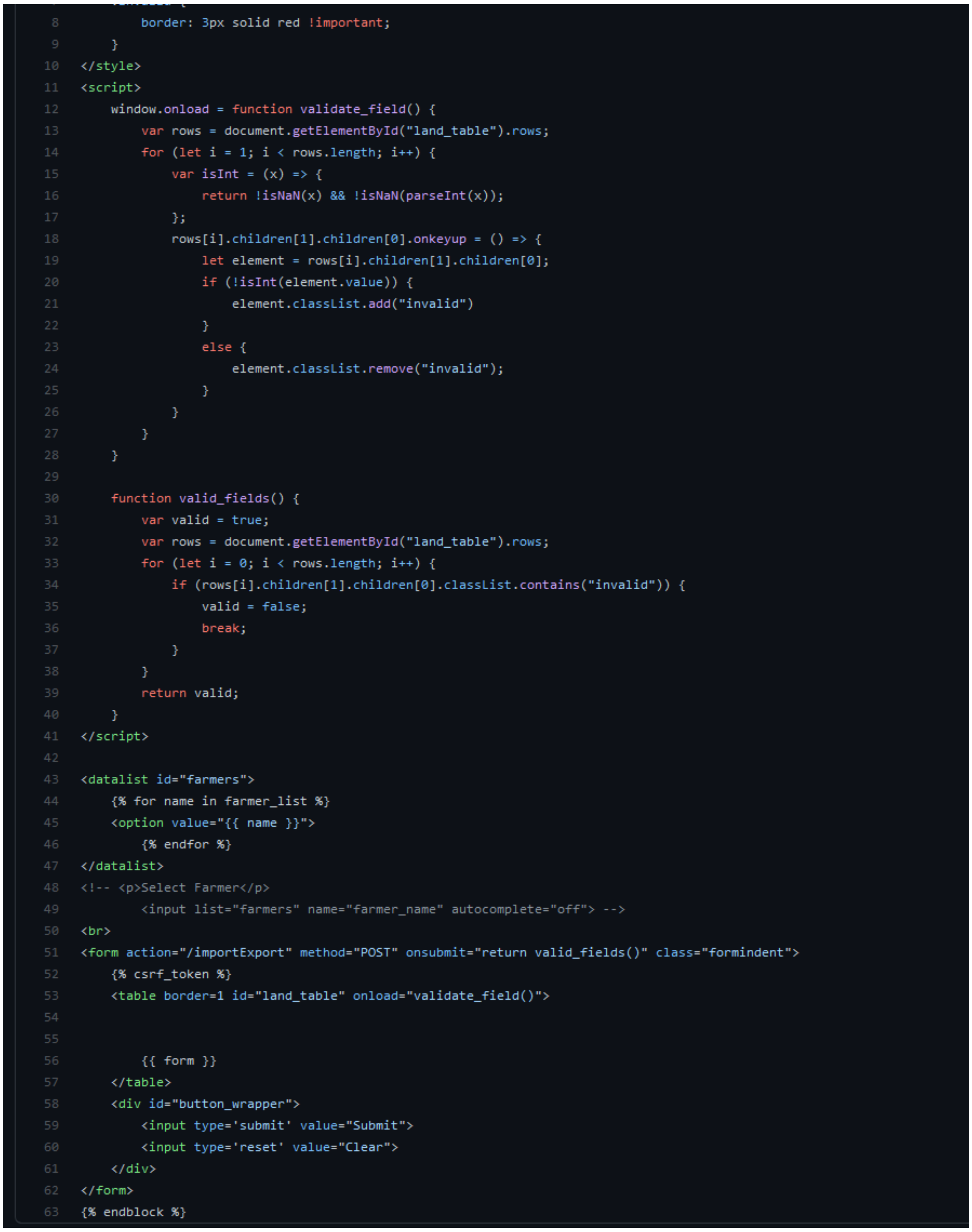
```
{% extends "base.html" %}
{% include "registration/login.html" %}

{% load static %}

{% block body %}
<style>
    .invalid {
        border: 3px solid red !important;
    }
</style>
<script>
    window.onload = function validate_field() {
        var elem = document.getElementById("land_table");
        let element = rows[i].children[3].children[0];
        if (!isInt(element.value)) {
            element.classList.add("invalid")
        }
        else {
            element.classList.remove("invalid");
        }

    function valid_fields() {
        var valid = true;
        var rows = document.getElementById("#id_herd_no");
        rows[i].children[3].children[0].onkeyup = () => {
            let element = rows[i].children[3].children[0];
            if (!isInt(element.value)) {
                element.classList.add("invalid")
            }
            else {
                element.classList.remove("invalid");
            }
        }
    }
</script>


<script>
    $(function () {
        $("#id_date").datepicker();
    });
</script>

<form action="/conductGrasslandAssessment" method="POST" onsubmit="return valid_fields()" class="formindent">
    {% csrf_token %}
    <table border=1 id="land_table" onload="validate_field()">
        {{ form }}

    </table>
    <div class="input-group-append" data-target="#datetimepicker1" data-toggle="datetimepicker">
        <div class="input-group-text"><i class="fa fa-calendar"></i></div>
    </div>
    </div>
    <div id="button_wrapper">
        <input type='submit' value="Submit">
        <input type='reset' value="Clear">
    </div>
</form>

{% endblock %}
```

Figure 28: Templates.py



```
{% extends "base.html" %}
{% include "registration/login.html" %}

{% block body %}
<style>
    td {
        text-align: center;
    }

    th {
        text-align: center;
    }
</style>
<form action="/grasslandAssessmentResult" method="POST" class="formindent">
    {% csrf_token %}
    <table border="1" id="table">
        <tr>
            <th scope="col">Total Nitrates</th>
            <th scope="col">Total Phosphates</th>
            <th scope="col">Total Land Area</th>
            <th scope="col">Grassland Stocking Rate</th>
            <th scope="col">Wholefarm Stocking Rate</th>
            <th scope="col">Livestock unit per Hectacre</th>
        </tr>
        {% for total in list_for_result %}
        <tr>
            <td>{{ total.0 }}</td>
            <td>{{ total.1 }}</td>
            <td>{{ total.2 }}</td>
            <td>{{ total.3 }}</td>
            <td>{{ total.4 }}</td>
            <td>{{ total.5 }}</td>
        </tr>
        {% endfor %}
    </table>
    <div id="button_wrapper">
        <a href="/" id="button_wrapper">
            <input type="button" value="Home" />
    </div>
</form>
{% endblock %}
```

Figure 29: Templates.py



```
        border: 3px solid red !important;
    }
</style>
<script>
    window.onload = function validate_field() {
        var rows = document.getElementById("land_table").rows;
        for (let i = 1; i < rows.length; i++) {
            var isInt = (x) => {
                return !isNaN(x) && !isNaN(parseInt(x));
            };
            rows[i].children[1].children[0].onkeyup = () => {
                let element = rows[i].children[1].children[0];
                if (!isInt(element.value)) {
                    element.classList.add("invalid")
                }
                else {
                    element.classList.remove("invalid");
                }
            }
        }
    }

    function valid_fields() {
        var valid = true;
        var rows = document.getElementById("land_table").rows;
        for (let i = 0; i < rows.length; i++) {
            if (rows[i].children[1].children[0].classList.contains("invalid")) {
                valid = false;
                break;
            }
        }
        return valid;
    }
</script>

<datalist id="farmers">
    {% for name in farmer_list %}
    <option value="{{ name }}">
        {% endfor %}
</datalist>
<!-- <p>Select Farmer</p>
        <input list="farmers" name="farmer_name" autocomplete="off"> -->
<br>
<form action="/importExport" method="POST" onsubmit="return valid_fields()" class="formindent">
    {% csrf_token %}
    <table border=1 id="land_table" onload="validate_field()">


        {{ form }}
    </table>
    <div id="button_wrapper">
        <input type='submit' value="Submit">
        <input type='reset' value="Clear">
    </div>
</form>
{% endblock %}
```

Figure 30: Templates.py