



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:

<b>Introduction</b> .....	4
<b>Table of Contents:</b> .....	2
<b>Table of Figures:</b> .....	3
<b>Project Code</b> .....	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

```
38
39 @login_required
40 @csrf_protect
41 def home(request):
42     # e = Exportation(exportation_original_stocking_rate = 15,
43     # export = 20, person_accepting_import = "Michael", new_stocking_rate = 20)
44     # e.save()
45     return TemplateResponse(request, "home.html")
46
47
48 @login_required
49 @csrf_protect
50 def conductGrasslandAssessment(request):
51     """
52     This function is responsible for storing the personal information associated with the farmer
53     """
54     if request.method == "POST":
55         # herdno_list = Farmer.objects.values_list("herd_no", flat=True)
56         form = GrasslandForm(request.POST)
57
58         # if(any( herd.herd_no == form['herd_no'].value() for herd in herdno_list )):
59         #     # It exists
60
61         # for herd in herdno_list:
62         #     if herd.herd_no == form['herd_no'].value():
63         #         # It exists
64
65         farmer = Farmer(
66             name=form["farmer_name"].value(),
67             address=form["farmer_address_line_1"].value()
68             + " "
69             + form["farmer_address_line_2"].value()
70             + " "
71             + form["farmer_address_line_3"].value(),
72             date=parse(form["date"].value(), dayfirst=True).strftime("%Y-%m-%d"),
73             county=form["county"].value(),
74             herd_no=form["herd_no"].value()
75         )
76
77         # if county in counties_with_attrs:
78         values = counties_with_attrs.get(form["county"].value().lower())
79
80         farmer.save()
81         request.session["farmer_id"] = farmer.id
82
83         return redirect("/conductGrasslandAssessment2")
84     return render(request, "conductGrasslandAssessment.html", {"form": GrasslandForm()})
85
86
87 def record5_calculations(owned, rented, time):
88     """
89     This function performs the record 5 calculation
90     """
91     time /= 12
92     rounded_time = round(time, 2)
93     rented = rented * time
94     rounded_rented = round(rented, 2)
95     owned += rounded_rented
96
97     return owned
98
99
```

Figure 1: Views.py

```

100 @login_required
101 @csrf_protect
102 def conductGrasslandAssessment2(request):
103     """
104     This function records the inputted land information, performs relevant calculations and
105     stores the data
106     """
107     if request.method == "POST":
108         form = Grassland2(request.POST)
109         if form.is_valid():
110             farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
111             landInfo = Grassland(
112                 farmer_id=farmer,
113                 owned_land=(owned := float(form["owned_land"].value())),
114                 rented_land=(rented := float(form["rented_land"].value())),
115                 time_rented=(time_r := int(form["time_rented"].value())),
116                 total_tillage_area=(
117                     tillage := float(form["total_tillage_area"].value())
118                 ),
119                 area_reseeded=float(form["area_reseeded"].value()),
120                 total_grass_area=(area := record5_calculations(owned, rented, time_r)),
121                 total_land_area=area + tillage,
122             )
123             landInfo.save()
124             request.session["grassland_id"] = landInfo.id
125             return redirect("/conductGrasslandAssessment5")
126         else:
127             return render(
128                 request, "conductGrasslandAssessment2.html", {"form": Grassland2()}
129             )
130
131     return render(request, "conductGrasslandAssessment2.html", {"form": Grassland2()})
132
133
134 @login_required
135 @csrf_protect
136 def conductGrasslandAssessment3(request):
137     """
138     This function records information relating to a fertilizer plan, this feature is
139     not yet implemented
140     """
141     if request.method == "POST":
142         form = Grassland3(request.POST)
143
144         grass = Grassland.objects.get(id=request.session.get("grassland_id"))
145
146         grass.sample_code = form["sample_code"].value()
147         grass.sample_area = form["sample_area"].value()
148         grass.date_taken = (
149             date := parse(form["date_taken"].value(), dayfirst=True)
150         ).strftime("%Y-%m-%d")
151         grass.expiry_date = date.replace(year=date.year + 5)
152         grass.ph = form["ph"].value()
153         grass.lime_required = form["lime_required"].value()
154         grass.p_value = form["p_value"].value()
155         grass.k_value = form["k_value"].value()
156         grass.save()
157
158     return redirect("/conductGrasslandAssessment4")
159     return render(request, "conductGrasslandAssessment3.html", {"form": Grassland3})
160

```

Figure 2: Views.py

```

162 @login_required
163 @csrf_protect
164 def conductGrasslandAssessment4(request):
165     """
166     This function records information relating to a fertilizer plan, the function will
167     record the amount of feed a farmer owns and calculates whether they are within limits.
168     this feature is
169     not yet implemented
170     """
171     if request.method == "POST":
172         form = Grassland4(request.POST)
173
174         # need to get a value to associate the feeds with for grassland table
175         # possibly feed tonnage
176         farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
177         grass = Grassland.objects.get(id=request.session.get("grassland_id"))
178
179         num_of_feed = Farmer_Feed(
180             farmer_id=farmer,
181             number_compound=(num1 := float(form["number_compound"].value())),
182             number_wheat=(num2 := float(form["number_wheat"].value())),
183             number_maize=(num3 := float(form["number_maize"].value())),
184             number_maize_germ=(num4 := float(form["number_maize_germ"].value())),
185             number_oats=(num5 := float(form["number_oats"].value())),
186             number_beat_pulps_molassed=(
187                 num6 := float(form["number_beat_pulps_molassed"].value())
188             ),
189             number_beat_pulp_unmolassed=(
190                 num7 := float(form["number_beat_pulp_unmolassed"].value())
191             ),
192             number_citrus_pulp=(num8 := float(form["number_citrus_pulp"].value())),
193             number_maize_distiller=(
194                 num9 := float(form["number_maize_distiller"].value())
195             ),
196             number_maize_gluten=(num10 := float(form["number_maize_gluten"].value())),
197             number_copra=(num11 := float(form["number_copra"].value())),
198             number_cotton_seed=(num12 := float(form["number_cotton_seed"].value())),
199             number_palm_kernel=(num13 := float(form["number_palm_kernel"].value())),
200             number_rapeseed=(num14 := float(form["number_rapeseed"].value())),
201             number_soya_bean=(num15 := float(form["number_soya_bean"].value())),
202             number_sunflower=(num16 := float(form["number_sunflower"].value())),
203             number_peas=(num17 := float(form["number_peas"].value())),
204             number_beans=(num18 := float(form["number_beans"].value())),
205             number_soya_hulls=(num19 := float(form["number_soya_hulls"].value())),
206             number_distillers_grain=(
207                 num20 := float(form["number_distillers_grain"].value())
208             ),
209             number_lucerne=(num21 := float(form["number_lucerne"].value())),
210         )
211         """
212         Calculating total tonnage. Not finished will be refactored
213         """
214         grass.feed_tonnage = (
215             total := (
216                 num1
217                 + num2
218                 + num3
219                 + num4
220                 + num5
221                 + num6
222                 + num7

```

Figure 3: Views.py

```

238     )
239     grass.save()
240     num_of_feed.save()
241     return redirect("/conductGrasslandAssessment5")
242     results = Feed_Types.objects.all()
243     form = Grassland4()
244     form = list(zip(results, form))
245     return render(request, "conductGrasslandAssessment4.html", {"form": form})
246
247
248 @login_required
249 @csrf_protect
250 def conductGrasslandAssessment5(request):
251     """
252     This function records a farmers livestock information
253     """
254     if request.method == "POST":
255         form = Grassland5(request.POST)
256
257         farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
258         grass = Grassland.objects.get(id=request.session.get("grassland_id"))
259         num_of_stock = Farmer_Livestock(
260             farmer_id=farmer,
261             number_dairy_cows=(num1 := float(form["number_dairy_cows"].value())),
262             number_suckler_cows=(num2 := float(form["number_suckler_cows"].value())),
263             number_cattle1=(num3 := float(form["number_cattle1"].value())),
264             number_cattle2=(num4 := float(form["number_cattle2"].value())),
265             number_cattle3=(num5 := float(form["number_cattle3"].value())),
266             number_mountain_ewe=(num6 := float(form["number_mountain_ewe"].value())),
267             number_lowland_ewe=(num7 := float(form["number_lowland_ewe"].value())),
268             number_mountain_hogget=(
269                 num8 := float(form["number_mountain_hogget"].value())
270             ),
271             number_lowland_hogget=(
272                 num9 := float(form["number_lowland_hogget"].value())
273             ),
274             number_goats=(num10 := float(form["number_goats"].value())),
275             number_horse1=(num11 := float(form["number_horse1"].value())),
276             number_horse2=(num12 := float(form["number_horse2"].value())),
277         )
278
279         animal_list = [
280             num1,
281             num2,
282             num3,
283             num4,
284             num5,
285             num6,
286             num7,
287             num8,
288             num9,
289             num10,
290             num11,
291             num12,
292         ]
293         """
294         Calculating total nitrates and phosphates from the input livestock figures
295         """
296         total_nitrates = 0
297         grass.number_of_animals = (total := (sum(animal_list)))
298         nitrate_results = Monthly_Livestock_Numbers.objects.values_list(
299             "organic_nitrates", flat=True
300         )

```

Figure 4: Views.py



```

292     ]
293     """
294     Calculating total nitrates and phosphates from the input livestock figures
295     """
296     total_nitrates = 0
297     grass.number_of_animals = (total := (sum(animal_list)))
298     nitrate_results = Monthly_Livestock_Numbers.objects.values_list(
299         "organic_nitrates", flat=True
300     )
301     for a, b in zip(animal_list, nitrate_results):
302         total_nitrates += a * b
303     grass.organicN = total_nitrates
304
305     total_potassium = 0
306     potass_results = Monthly_Livestock_Numbers.objects.values_list(
307         "organic_potassium", flat=True
308     )
309     for c, d in zip(animal_list, potass_results):
310         total_potassium += c * d
311     grass.organicP = total_potassium
312
313     """
314     Calculating livestock unit per hectare
315     """
316     total_lsu = 0
317     lsu_vals = Monthly_Livestock_Numbers.objects.values_list("lsu", flat=True)
318     for a, b in zip(animal_list, lsu_vals):
319         total_lsu += a * b
320     grass.lsu = total_lsu
321
322     num_of_stock.save()
323     grass.save()
324     return redirect("/grasslandReport")
325     results = Monthly_Livestock_Numbers.objects.all()
326     form = Grassland5()
327
328     form = list(zip(results, form))
329     return render(request, "conductGrasslandAssessment5.html", {"form": form})
330
331
332 @login_required
333 @csrf_protect
334 def grasslandAssessmentResult(request):
335     """
336     Displaying the results from the conduct assessment
337     """
338     everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
339     list_for_result = []
340     objects_to_update = []
341     for row in everything:
342         total_organic_n = row.organicN
343         total_organic_p = row.organicP
344         total_land_area = row.total_land_area
345         total_grass_area = row.total_grass_area
346         total_lsu = row.lsu
347
348         gsr = total_organic_n / total_grass_area
349         wfsr = total_organic_n / total_land_area
350
351         row.grassland_stocking_rate = gsr

```

Figure 5: Views.py

```

334 def grasslandAssessmentResult(request):
335     """
336     Displaying the results from the conduct assessment
337     """
338     everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
339     list_for_result = []
340     objects_to_update = []
341     for row in everything:
342         total_organic_n = row.organicN
343         total_organic_p = row.organicP
344         total_land_area = row.total_land_area
345         total_grass_area = row.total_grass_area
346         total_lsu = row.lsu
347
348         gsr = total_organic_n / total_grass_area
349         wfsr = total_organic_n / total_land_area
350
351         row.grassland_stocking_rate = gsr
352         row.wholefarm_stocking_rate = wfsr
353         objects_to_update.append(row)
354         list_for_result.append(
355             (
356                 total_organic_n,
357                 total_organic_p,
358                 total_land_area,
359                 round(gsr, 2),
360                 round(wfsr, 2),
361                 round(total_lsu, 2),
362             )
363         )
364
365     # The objects_to_update list will these columns in the database
366     Grassland.objects.bulk_update(
367         objects_to_update, ["grassland_stocking_rate", "wholefarm_stocking_rate"]
368     )
369     Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
370     return render(request, "grasslandReport.html", {"list_for_result": list_for_result})
371
372
373 @login_required
374 @csrf_protect
375 def importExport(request):
376     """
377     Taking in the import/export information
378     """
379     if request.method == "POST":
380         form = import_Export(request.POST)
381         try:
382             farmer_name = form["farmer_name"].value()
383             herd_no = farmer_name.split("-")[1].strip()
384             farmer = Farmer.objects.get(herd_no=herd_no)
385             if farmer == None:
386                 raise Exception()
387             request.session["farmer_id"] = farmer.id
388         except:
389             farmer_list = Farmer.objects.filter(is_assessed=True)
390             farmer_list = [
391                 f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
392             ]
393             return render(
394                 request,
395                 "importExport.html",
396                 {"form": import_Export, "farmer_list": farmer_list},

```

Figure 6: Views.py

```

395         importExport.html ,
396         {"form": import_Export, "farmer_list": farmer_list},
397     )
398
399     grass = Grassland.objects.get(farmer_id=request.session.get("farmer_id"))
400     farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
401
402     total_n = grass.organicN
403     area = grass.total_land_area
404     """
405     Selecting which action
406     """
407     if form["option"].value() == "Import":
408         farmer_import = Importation(
409             farmer_id=farmer,
410             farmyard_manure=(manure := int(form["farmyard_manure"].value())),
411             slurry=(slurry := int(form["slurry"].value())),
412             nitrates=(nit := int((slurry * 5) + manure * 4.5)),
413         )
414         total_n += nit
415         grass.organicN = total_n
416         orgN = grass.organicN
417         orgN / area
418         farmer_import.save()
419         grass.save()
420
421     elif form["option"].value() == "Export":
422         farmer_export = Exportation(
423             farmer_id=farmer,
424             farmyard_manure=(manure := int(form["farmyard_manure"].value())),
425             slurry=(slurry := int(form["slurry"].value())),
426             nitrates=(nit := int((slurry * 5) + manure * 4.5)),
427         )
428         total_n -= nit
429         grass.organicN = total_n
430         orgN = grass.organicN
431         orgN / area
432         grass.save()
433         farmer_export.save()
434
435     return redirect("/importExportReport")
436     farmer_list = Farmer.objects.filter(is_assessed=True)
437     farmer_list = [{"farmer.name"} - {"farmer.herd_no"} for farmer in farmer_list]
438     return render(
439         request,
440         "importExport.html",
441         {"form": import_Export, "farmer_list": farmer_list},
442     )
443
444
445 @login_required
446 @csrf_protect
447 def importExportReport(request):
448     """
449     Import Export Result
450     """
451     everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
452     list_for_result = []
453     objects_to_update = []
454     for row in everything:
455         total_organic_n = row.organicN
456         total_organic_p = row.organicP
457         total_land_area = row.total_land_area

```

Figure 7: Views.py

```

472         round(total_lsu, 2),
473     )
474 )
475
476 # The objects_to_update list will these columns in the database
477 Grassland.objects.bulk_update(
478     objects_to_update, ["grassland_stocking_rate", "wholefarm_stocking_rate"]
479 )
480 Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
481 return render(
482     request, "importExportReport.html", {"list_for_result": list_for_result}
483 )
484
485
486 @login_required
487 @csrf_protect
488 def storage_process(request):
489     """
490     method recording information relating to storage facilities
491     """
492     if request.method == "POST":
493         form = storage(request.POST)
494         try:
495             farmer_name = form["farmer_name"].value()
496             herd_no = farmer_name.split("-")[1].strip()
497             farmer = Farmer.objects.get(herd_no=herd_no)
498             if farmer == None:
499                 raise Exception()
500             request.session["farmer_id"] = farmer.id
501         except:
502             farmer_list = Farmer.objects.filter(is_assessed=True)
503             farmer_list = [
504                 f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
505             ]
506             return render(
507                 request,
508                 "storage.html",
509                 {"form": storage_process, "farmer_list": farmer_list},
510             )
511         county_val = Farmer.objects.get(
512             id=request.session.get("farmer_id")
513         ).county.lower()
514         rainfall_val = float(counties_with_attrs[county_val][1])
515         total_weeks = float(counties_with_attrs[county_val][3])
516         num_animals = Farmer_Livestock.objects.latest("farmer_id_id")
517         num_animals = model_to_dict(num_animals)
518         num_animals.pop("id")
519         num_animals.pop("farmer_id")
520         num_animals = dict(num_animals).values()
521         slurry_vals = list(
522             Monthly_Livestock_Numbers.objects.values_list("slurry_m3", flat=True)
523         )
524         manure_vals = list(
525             Monthly_Livestock_Numbers.objects.values_list("manure_m3", flat=True)
526         )
527
528     """
529     Retrieving values from tables
530     """
531     if form["choice"].value() == storage.TYPE[0][0]:
532         manure = sum((a * b for (a, b) in zip(num_animals, slurry_vals)))
533         lengt = (leng := float(form["length"].value()))
534         lengt -= 0.3
535         bread = (breth := float(form["breadth"].value()))

```

Figure 8: Views.py

```

589         total_storage=total_storage,
590         total_slurry_manure=manure,
591         space_available=space_available,
592         max_storage=req_storage,
593     )
594     if form["add_another_container"].value():
595         if "Storagelist" in request.session:
596             request.session["Storagelist"].append(model_to_dict(storage_form))
597             request.session.modified = True
598         else:
599             request.session["Storagelist"] = [model_to_dict(storage_form)]
600             farmer_list = Farmer.objects.filter(is_assessed=True)
601             farmer_list = [
602                 f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
603             ]
604             return render(
605                 request, "storage.html", {"form": storage, "farmer_list": farmer_list}
606             )
607     else:
608         save_list = request.session.get("Storagelist", [])
609         save_list.append(model_to_dict(storage_form))
610         initial = 0 if slurry_object == None else slurry_object.total_storage
611         storage_list = []
612         for shed in save_list:
613             storage_form = Slurry_Storage(
614                 farmer_id=farmer,
615                 length=shed["length"],
616                 rainfall=shed["rainfall"],
617                 total_storage=shed["total_storage"],
618                 breadth=shed["breadth"],
619                 height=shed["height"],
620                 total_slurry_manure=shed["total_slurry_manure"],
621                 space_available=shed["space_available"],
622                 max_storage=shed["max_storage"],
623             )
624             storage_list.append(storage_form)
625             if "Storagelist" in request.session:
626                 del request.session["Storagelist"]
627             Slurry_Storage.objects.bulk_create(storage_list)
628
629         return redirect("storage_report")
630
631     farmer_list = Farmer.objects.filter(is_assessed=True)
632     farmer_list = [f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list]
633     return render(
634         request, "storage.html", {"form": storage, "farmer_list": farmer_list}
635     )
636
637
638 @login_required
639 @csrf_protect
640 def storage_report(request):
641     """
642     displaying the results from storage function
643     """
644     everything = Slurry_Storage.objects.filter(
645         farmer_id=request.session.get("farmer_id")
646     )
647     list_for_result = []
648     objects_to_update = []
649     county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
650     for row in everything:
651         max_storage = row.max_storage
652         total_storage = row.total_storage

```

Figure 9: Views.py

```

543         space_available = total_storage - req_storage
544         space_available = round(space_available, 2)
545
546         elif form["option"].value() == storage.CHOICES[1][0]:
547             total_storage = lengt * bread * heigh
548             deduction = rainfall_val * total_weeks
549             deduction = round(deduction, 2)
550             total_storage -= deduction
551             space_available = total_storage - req_storage
552             space_available = round(space_available, 2)
553
554         elif form["choice"].value() == storage.TYPE[1][0]:
555             manure = sum((a * b for (a, b) in zip(num_animals, manure_vals)))
556             lengt = (lengt := float(form["length"].value()))
557             bread = (breth := float(form["breadth"].value()))
558             heigh = (hight := float(form["height"].value()))
559             req_storage = manure * total_weeks
560
561         if form["option"].value() == storage.CHOICES[0][0]:
562             total_storage = lengt * bread * heigh
563             space_available = total_storage - req_storage
564             space_available = round(space_available, 2)
565
566         elif form["option"].value() == storage.CHOICES[1][0]:
567             total_storage = lengt * bread * heigh
568             deduction = rainfall_val * total_weeks
569             deduction = round(deduction, 2)
570             total_storage -= deduction
571             space_available = total_storage - req_storage
572             space_available = round(space_available, 2)
573
574         manure = round(manure, 2)
575         total_storage = round(total_storage, 2)
576         farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
577         try:
578             slurry_object = Slurry_Storage.objects.filter(
579                 farmer_id_id=request.session.get("farmer_id")
580             ).latest("id")
581         except:
582             slurry_object = None
583         storage_form = Slurry_Storage(
584             farmer_id=farmer,
585             length=lengt,
586             breadth=breath,
587             height=heigh,
588             rainfall=rainfall_val,
589             total_storage=total_storage,
590             total_slurry_manure=manure,
591             space_available=space_available,
592             max_storage=req_storage,
593         )
594         if form["add_another_container"].value():
595             if "Storagelist" in request.session:
596                 request.session["Storagelist"].append(model_to_dict(storage_form))
597                 request.session.modified = True
598             else:
599                 request.session["Storagelist"] = [model_to_dict(storage_form)]
600             farmer_list = Farmer.objects.filter(is_assessed=True)
601             farmer_list = [
602                 f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
603             ]
604             return render(
605                 request, "storage.html", {"form": storage, "farmer_list": farmer_list}
606             )

```

Figure 10: Views.py

```

635 )
636
637
638 @login_required
639 @csrf_protect
640 def storage_report(request):
641     """
642     displaying the results from storage function
643     """
644     everything = Slurry_Storage.objects.filter(
645         farmer_id=request.session.get("farmer_id")
646     )
647     list_for_result = []
648     objects_to_update = []
649     county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
650     for row in everything:
651         max_storage = row.max_storage
652         total_storage = row.total_storage
653
654         space_available = row.space_available
655         total_slurry_manure = row.total_slurry_manure
656
657         objects_to_update.append(row)
658         list_for_result.append(
659             (
660                 county_val,
661                 total_slurry_manure,
662                 total_storage,
663                 round(max_storage, 2),
664                 space_available,
665             )
666         )
667
668     # The objects_to_update list will have these columns in the database
669     Slurry_Storage.objects.bulk_update(
670         objects_to_update,
671         ["total_slurry_manure", "total_storage", "max_storage", "space_available"],
672     )
673     Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
674     return render(request, "storage_report.html", {"list_for_result": list_for_result})
675
676
677 @login_required
678 @csrf_protect
679 def update_lsu(request):
680     """
681     Feature for retrieving farmers current number livestock, feature will change
682     lsu and nitrates when livestock figures are updated
683     """
684     if request.session.get("farmer_id") is None:
685         return redirect("home")
686     else :
687
688         if request.method == "POST":
689             form = storage(request.POST)
690             try:
691                 farmer_name = form["farmer_name"].value()
692                 herd_no = farmer_name.split("-")[1].strip()
693                 farmer = Farmer.objects.get(herd_no=herd_no)
694                 if farmer == None:
695                     raise Exception()
696                 request.session["farmer_id"] = farmer.id
697             except:
698                 farmer_list = Farmer.objects.filter(is_assessed=True)

```

Figure 11: Views.py

```

676
677 @login_required
678 @csrf_protect
679 def update_lsu(request):
680     """
681     Feature for retrieving farmers current number livestock, feature will change
682     lsu and nitrates when livestock figures are updated
683     """
684     if request.session.get("farmer_id") is None:
685         return redirect("home")
686     else :
687
688         if request.method == "POST":
689             form = storage(request.POST)
690             try:
691                 farmer_name = form["farmer_name"].value()
692                 herd_no = farmer_name.split("-")[1].strip()
693                 farmer = Farmer.objects.get(herd_no=herd_no)
694                 if farmer == None:
695                     raise Exception()
696                 request.session["farmer_id"] = farmer.id
697             except:
698                 farmer_list = Farmer.objects.filter(is_assessed=True)
699                 farmer_list = [
700                     f"{farmer.name} - {farmer.herd_no}" for farmer in farmer_list
701                 ]
702                 return render(
703                     request,
704                     "conductGrasslandAssessment5.html",
705                     {"form": Grassland5, "farmer_list": farmer_list},
706                 )
707
708                 farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
709                 grass = Grassland.objects.get(id=request.session.get("grassland_id"))
710                 num_of_stock = Farmer_Livestock(
711                     farmer_id=farmer,
712                     number_dairy_cows=(num1 := float(form["number_dairy_cows"].value())),
713                     number_suckler_cows=(num2 := float(form["number_suckler_cows"].value())),
714                     number_cattle1=(num3 := float(form["number_cattle1"].value())),
715                     number_cattle2=(num4 := float(form["number_cattle2"].value())),
716                     number_cattle3=(num5 := float(form["number_cattle3"].value())),
717                     number_mountain_ewe=(num6 := float(form["number_mountain_ewe"].value())),
718                     number_lowland_ewe=(num7 := float(form["number_lowland_ewe"].value())),
719                     number_mountain_hogget=(
720                         num8 := float(form["number_mountain_hogget"].value())
721                     ),
722                     number_lowland_hogget=(
723                         num9 := float(form["number_lowland_hogget"].value())
724                     ),
725                     number_goats=(num10 := float(form["number_goats"].value())),
726                     number_horse1=(num11 := float(form["number_horse1"].value())),
727                     number_horse2=(num12 := float(form["number_horse2"].value())),
728                 )
729
730                 animal_list = [
731                     num1,
732                     num2,
733                     num3,
734                     num4,
735                     num5,
736                     num6,
737                     num7,
738                     num8,
739                     num9,

```

Figure 12: Views.py



```

747         organic_nitrates, flat=True
748     )
749     for a, b in zip(animal_list, nitrate_results):
750         total_nitrates += a * b
751     grass.organicN = total_nitrates
752
753     total_potassium = 0
754     potass_results = Monthly_Livestock_Numbers.objects.values_list(
755         "organic_potassium", flat=True
756     )
757     for c, d in zip(animal_list, potass_results):
758         total_potassium += c * d
759     grass.organicP = total_potassium
760
761     total_lsu = 0
762     lsu_vals = Monthly_Livestock_Numbers.objects.values_list("lsu", flat=True)
763     for a, b in zip(animal_list, lsu_vals):
764         total_lsu += a * b
765     grass.lsu = total_lsu
766
767     num_of_stock.save()
768     grass.save()
769     return redirect("/grasslandReport")
770
771     farmer = Farmer.objects.get(id=request.session.get("farmer_id"))
772     number_animals = Farmer_Livestock.objects.filter(farmer_id=farmer.id).latest("id")
773     d = model_to_dict(number_animals)
774     d.pop("id")
775     d.pop("farmer_id")
776     form = Grassland5(initial=d)
777     form = list(zip(Monthly_Livestock_Numbers.objects.all(), form))
778     return render(request, "conductGrasslandAssessment5.html", {"form": form})
779
780
781 @login_required
782 @csrf_protect
783 def view_records(request):
784     """
785     Full report
786     """
787     if request.session.get("farmer_id") is None:
788         return redirect("home")
789     else :
790
791         everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
792         slurrn = Slurry_Storage.objects.filter(farmer_id=request.session.get("farmer_id"))
793         county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
794         list_for_result = []
795         objects_to_update = []
796         for row in everything:
797             total_organic_n = row.organicN
798             total_organic_p = row.organicP
799             total_land_area = row.total_land_area
800             total_grass_area = row.total_grass_area
801             total_lsu = row.lsu
802
803             gsr = total_organic_n / total_grass_area
804             wfsr = total_organic_n / total_land_area
805
806             row.grassland_stocking_rate = gsr
807             row.wholefarm_stocking_rate = wfsr
808             objects_to_update.append(row)
809             list_for_result.append(
810                 [

```

Figure 13: Views.py

```

782 @csrf_protect
783 def view_records(request):
784     """
785     Full report
786     """
787     if request.session.get("farmer_id") is None:
788         return redirect("home")
789     else :
790
791         everything = Grassland.objects.filter(farmer_id=request.session.get("farmer_id"))
792         slurr = Slurry_Storage.objects.filter(farmer_id=request.session.get("farmer_id"))
793         county_val = Farmer.objects.get(id=request.session.get("farmer_id")).county
794         list_for_result = []
795         objects_to_update = []
796         for row in everything:
797             total_organic_n = row.organicN
798             total_organic_p = row.organicP
799             total_land_area = row.total_land_area
800             total_grass_area = row.total_grass_area
801             total_lsu = row.lsu
802
803             gsr = total_organic_n / total_grass_area
804             wfsr = total_organic_n / total_land_area
805
806             row.grassland_stocking_rate = gsr
807             row.wholefarm_stocking_rate = wfsr
808             objects_to_update.append(row)
809             list_for_result.append(
810                 [
811                     total_organic_n,
812                     total_organic_p,
813                     total_land_area,
814                     round(gsr, 2),
815                     round(wfsr, 2),
816                     round(total_lsu, 2),
817                 ]
818             )
819
820         for rows in slurr:
821             max_storage = rows.max_storage
822             total_storage = rows.total_storage
823
824             space_available = rows.space_available
825             total_slurry_manure = rows.total_slurry_manure
826
827             objects_to_update.append(rows)
828             list_for_result[-1] = list_for_result[-1] + [
829                 county_val,
830                 total_slurry_manure,
831                 total_storage,
832                 round(max_storage, 2),
833                 space_available,
834             ]
835
836         # The objects_to_update list will these columns in the database
837         Farmer.objects.filter(id=request.session.get("farmer_id")).update(is_assessed=True)
838         return render(request, "records.html", {"list_for_result": list_for_result})

```

Figure 14: Views.py

## Models.py

```
93
94 class Farmer(models.Model):
95     """
96     This is the farmer model, the attributes to be stored in the database
97     """
98     name = models.CharField(max_length=30)
99     address = models.CharField(max_length=30)
100    county = models.CharField(max_length=30, choices=counties, null=True)
101    date = models.DateField(null=True)
102    herd_no = models.CharField(max_length=30, null=True, unique=True)
103    is_assessed = models.IntegerField(null=True, default=0)
104    zone = models.CharField(max_length=30)
105
106
107 class Grassland(models.Model):
108     """
109     This is the Grassland model, the attributes to be stored in the database.
110     This will be changing after implementation of a new feature
111     """
112    farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
113    owned_land = models.FloatField(null=True)
114    rented_land = models.FloatField(null=True)
115    time_rented = models.IntegerField(null=True)
116    total_grass_area = models.FloatField(null=True)
117    total_tillage_area = models.FloatField(null=True)
118    total_land_area = models.FloatField(null=True)
119    area_reseeded = models.FloatField(null=True)
120    organicN = models.FloatField(null=True)
121    organicP = models.FloatField(null=True)
122    type_of_stock = models.CharField(max_length=30, null=True)
123    type_of_feed = models.CharField(max_length=30, null=True)
124    feed_name = models.CharField(max_length=30, null=True)
125    feed_tonnage = models.CharField(max_length=30, null=True)
126    number_of_animals = models.CharField(max_length=30, null=True)
127    grassland_stocking_rate = models.FloatField(null=True)
128    wholefarm_stocking_rate = models.FloatField(null=True)
129    imports = models.FloatField(null=True)
130    exports = models.FloatField(null=True)
131    legalN_limit = models.FloatField(null=True)
132    legalP_limit = models.FloatField(null=True)
133    lsu = models.FloatField(null=True)
134
135    # concentrateFed = models.FloatField(null=True)
136    # soil_samples = models.CharField(max_length=30)
137    # reseeding = models.CharField(max_length=30)
138    # lime_required = models.FloatField(null=True)
139    # enterprise = models.CharField(max_length=30)
140    # sample_code = models.CharField(max_length=30, null=True)
141    # date_taken = models.DateField(null=True)
142    # expiry_date = models.DateField(null=True)
143    # sample_area = models.FloatField(null=True)
144    # ph = models.FloatField(null=True)
145    # lime_required = models.FloatField(null=True)
146    # p_value = models.FloatField(null=True)
147    # p_index = models.FloatField(null=True)
148    # k_value = models.FloatField(null=True)
149    # k_index = models.FloatField(null=True)
150
```

Figure 15: Models.py

```

149 # k_index = models.FloatField(null=True)
150
151 class Importation(models.Model):
152     """
153     This is the Importation model, the attributes to be stored in the database.
154     """
155     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
156     farmyard_manure = models.IntegerField(null=True)
157     slurry = models.IntegerField(null=True)
158     nitrates = models.IntegerField(null=True)
159     phosphate = models.IntegerField(null=True)
160
161
162 class Exportation(models.Model):
163     """
164     This is the Exportation model, the attributes to be stored in the database.
165     """
166     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
167     farmyard_manure = models.IntegerField(null=True)
168     slurry = models.IntegerField(null=True)
169     nitrates = models.IntegerField(null=True)
170     phosphate = models.IntegerField(null=True)
171
172
173 class Monthly_Livestock_Numbers(models.Model):
174     """
175     This is the Monthly livestock numbers model, the attributes to be stored
176     in the database. This model also contains important information relating to
177     each breed of livestock
178     """
179     monthly_livestock_numbers = models.TextField(null=True)
180     type_of_animal = models.CharField(max_length=30)
181     organic_nitrates = models.FloatField(null=True)
182     organic_potassium = models.FloatField(null=True)
183     lsu = models.FloatField(null=True)
184     slurry_m3 = models.FloatField(null=True)
185     manure_m3 = models.FloatField(null=True)
186
187
188 # class Tillage(models.Model):
189     """
190     This is the Feed Types model, not currently implemented.
191     """
192     # tillage_year = models.DateTimeField(null=True)
193     # tillage_imports = models.FloatField(null=True)
194     # area_tillage = models.FloatField(null=True)
195     # area_grassland = models.FloatField(null=True)
196     # organic_nitrates_applied = models.FloatField(null=True)
197     # organic_phosphorus_applied = models.FloatField(null=True)
198     # applied_potassium = models.FloatField(null=True)
199     # field = models.CharField(max_length=30)
200     # fertilizer_allowed = models.FloatField(null=True)
201
202
203 # class Fertilizer_Plan(models.Model):
204     """
205     This is the Feed Types model, not currently implemented.
206     """
207     # opening_phosphorus = models.FloatField(null=True)
208     # opening_nitrogen = models.FloatField(null=True)
209     # opening_stock = models.CharField(max_length=30)
210     # planned_purchases = models.CharField(max_length=30)
211     # lime = models.FloatField(null=True)

```

Figure 16: Models.py

```

215
216 class Slurry_Storage(models.Model):
217     """
218     This is the Storage model, the attributes to be stored in the database.
219     This will contain the dimensions of the storage containers
220     """
221     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
222     length = models.FloatField(null=True)
223     breadth = models.FloatField(null=True)
224     height = models.FloatField(null=True)
225     total_slurry_manure = models.FloatField(null=True)
226     total_storage = models.FloatField(null=True)
227     rainfall = models.FloatField(null=True)
228     max_storage = models.FloatField(null=True)
229     space_available = models.FloatField(null=True)
230
231
232 class Farm_Records(models.Model):
233     """
234     This is the Farm Records model, the attributes to be stored in the database.
235     This model is not currently in use until implementation of new feature
236     """
237     farm_records_year = models.DateTimeField(null=True)
238     farm_records_max_nitrogen_allowed = models.FloatField(null=True)
239     farm_records_max_phospheros_allowed = models.FloatField(null=True)
240     farm_records_opening_stock = models.CharField(max_length=30)
241     fertilizer_bought = models.CharField(max_length=30)
242     fertilizer_sold = models.CharField(max_length=30)
243     closing_stock = models.CharField(max_length=30)
244     total_usage = models.FloatField(null=True)
245     balance_under_recommended = models.FloatField(null=True)
246     balance_under_legal_amount = models.FloatField(null=True)
247     import_export_information = models.CharField(max_length=30)
248
249
250 class Farmer_Livestock(models.Model):
251     """
252     This is the Farmer Livestock model, the attributes to be stored in the database.
253     This model will store the amount of livestock a farmer owns
254     """
255     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
256     number_dairy_cows = models.IntegerField(null=True)
257     number_suckler_cows = models.IntegerField(null=True)
258     number_cattle1 = models.IntegerField(null=True)
259     number_cattle2 = models.IntegerField(null=True)
260     number_cattle3 = models.IntegerField(null=True)
261     number_mountain_ewe = models.IntegerField(null=True)
262     number_lowland_ewe = models.IntegerField(null=True)
263     number_mountain_hogget = models.IntegerField(null=True)
264     number_lowland_hogget = models.IntegerField(null=True)
265     number_goats = models.IntegerField(null=True)
266     number_horse1 = models.IntegerField(null=True)
267     number_horse2 = models.IntegerField(null=True)
268
269
270 class Feed_Types(models.Model):
271     """
272     This is the Feed Types model, not currently implemented.
273     """
274     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
275     feed_type = models.CharField(max_length=30)
276     feed_name = models.CharField(max_length=40)
277     kg_per_tonne_fed = models.FloatField(null=True)

```

Figure 17: Models.py

```

250 class Farmer_Livestock(models.Model):
251     """
252     This is the Farmer Livestock model, the attributes to be stored in the database.
253     This model will store the amount of livestock a farmer owns
254     """
255     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
256     number_dairy_cows = models.IntegerField(null=True)
257     number_suckler_cows = models.IntegerField(null=True)
258     number_cattle1 = models.IntegerField(null=True)
259     number_cattle2 = models.IntegerField(null=True)
260     number_cattle3 = models.IntegerField(null=True)
261     number_mountain_ewe = models.IntegerField(null=True)
262     number_lowland_ewe = models.IntegerField(null=True)
263     number_mountain_hogget = models.IntegerField(null=True)
264     number_lowland_hogget = models.IntegerField(null=True)
265     number_goats = models.IntegerField(null=True)
266     number_horse1 = models.IntegerField(null=True)
267     number_horse2 = models.IntegerField(null=True)
268
269
270 class Feed_Types(models.Model):
271     """
272     This is the Feed Types model, not currently implemented.
273     """
274     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
275     feed_type = models.CharField(max_length=30)
276     feed_name = models.CharField(max_length=40)
277     kg_per_tonne_fed = models.FloatField(null=True)
278
279
280 class Farmer_Feed(models.Model):
281     """
282     This is the Feed Types model, not currently implemented.
283     """
284     farmer_id = models.ForeignKey(Farmer, on_delete=models.CASCADE, default=1)
285     number_compound = models.IntegerField(null=True)
286     number_wheat = models.IntegerField(null=True)
287     number_maize = models.IntegerField(null=True)
288     number_maize_germ = models.IntegerField(null=True)
289     number_oats = models.IntegerField(null=True)
290     number_beat_pulps_molassed = models.IntegerField(null=True)
291     number_beat_pulp_unmolassed = models.IntegerField(null=True)
292     number_citrus_pulp = models.IntegerField(null=True)
293     number_maize_distiller = models.IntegerField(null=True)
294     number_maize_gluten = models.IntegerField(null=True)
295     number_copra = models.IntegerField(null=True)
296     number_cotton_seed = models.IntegerField(null=True)
297     number_palm_kernel = models.IntegerField(null=True)
298     number_rapeseed = models.IntegerField(null=True)
299     number_soya_bean = models.IntegerField(null=True)
300     number_sunflower = models.IntegerField(null=True)
301     number_peas = models.IntegerField(null=True)
302     number_beans = models.IntegerField(null=True)
303     number_soya_hulls = models.IntegerField(null=True)
304     number_distillers_grain = models.IntegerField(null=True)
305     number_lucerne = models.IntegerField(null=True)

```

Figure 18: Models.py

## Tests

```
5 def test_web(client):
6     url = reverse("importExport")
7     response = client.get(url)
8     assert response.status_code == 302
9
10
11 def test_grass_stock_rate_1():
12     dairy_cow_nitrates = 89 * 10
13     cattle_1_nitrates = 65 * 15
14     cattle_2_nitrates = 57 * 4
15
16     total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
17     total_nitrates /= 20
18     total_nitrates = round(total_nitrates, 2)
19
20     assert total_nitrates == 104.65
21
22 def test_grass_stock_rate_2():
23     dairy_cow_nitrates = 89 * 30
24     cattle_1_nitrates = 65 * 55
25     cattle_2_nitrates = 57 * 40
26
27     total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
28     total_nitrates /= 20
29     total_nitrates = round(total_nitrates, 2)
30
31     assert total_nitrates == 426.25
32
33 def test_wholefarm_stocking_rate_1():
34     dairy_cow_nitrates = 10 * 89
35     cattle_1_nitrates = 15 * 65
36     cattle_2_nitrates = 4 * 57
37
38     total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
39     total_nitrates /= 23
40     wfsr = round(total_nitrates, 2)
41
42     assert wfsr == 91
43
44 def test_wholefarm_stocking_rate_2():
45     dairy_cow_nitrates = 50 * 89
46     cattle_1_nitrates = 35 * 65
47     cattle_2_nitrates = 24 * 57
48
49     total_nitrates = dairy_cow_nitrates + cattle_1_nitrates + cattle_2_nitrates
50     total_nitrates /= 43
51     wfsr = round(total_nitrates, 2)
52
53     assert wfsr == 188.21
54
55 def test_livestock_unit_hectacre_1():
56     dairy_lsu = 10 * 1
57     cattle_1_lsu = 15 * 1
58     cattle_2_lsu = 4 * 0.4
59
60     total_lsu = dairy_lsu + cattle_1_lsu + cattle_2_lsu
61     total_lsu /= 23
62     total_lsu = round(total_lsu, 2)
63
64     assert total_lsu == 1.16
```

Figure 19: Test.py

```

66 def test_livestock_unit_hectacre_2():
67     dairy_lsu = 30 * 1
68     cattle_1_lsu = 25 * 1
69     cattle_2_lsu = 14 * 0.4
70
71     total_lsu = dairy_lsu + cattle_1_lsu + cattle_2_lsu
72     total_lsu /= 23
73     total_lsu = round(total_lsu,2)
74
75     assert total_lsu == 2.63
76
77 def test_record5_1():
78     hectares = 10
79     time_r = 6
80     nitrates = 2093
81     land = 20
82
83     time_r /= 12
84     time_r = round(time_r,2)
85     hectares *= time_r
86     hectares = round(hectares,2)
87
88     gsr = nitrates / land
89     gsr = round(gsr,2)
90     assert gsr == 104.65
91
92     hectares += land
93     gsr = nitrates / hectares
94     gsr = round(gsr,2)
95     assert gsr == 83.72
96
97 def test_record5_2():
98     hectares = 8
99     time_r = 4
100    nitrates = 2093
101    land = 30
102
103    time_r /= 12
104    time_r = round(time_r,2)
105    hectares *= time_r
106    hectares = round(hectares,2)
107
108    gsr = nitrates / land
109    gsr = round(gsr,2)
110    assert gsr == 69.77
111
112    hectares += land
113    gsr = nitrates / hectares
114    gsr = round(gsr,2)
115    assert gsr == 64.12
116
117 def test_import_slurry_nitrates_1():
118     total_nitrates = 2093
119     import_ = 10
120     land = 23
121     nitrate_import = import_ * 5
122
123     total_nitrates += nitrate_import
124     wfsr = total_nitrates / land
125     wfsr = round(wfsr,2)

```

Figure 20: Test.py



```

118     total_nitrates = 2093
119     import_ = 10
120     land = 23
121     nitrate_import = import_ * 5
122
123     total_nitrates += nitrate_import
124     wfsr = total_nitrates / land
125     wfsr = round(wfsr,2)
126
127     assert wfsr == 93.17
128
129 def test_import_slurry_nitrates_2():
130     total_nitrates = 2093
131     import_ = 20
132     land = 43
133     nitrate_import = import_ * 5
134
135     total_nitrates += nitrate_import
136     wfsr = total_nitrates / land
137     wfsr = round(wfsr,2)
138
139     assert wfsr == 51.0
140
141 def test_import_manure_nitrates_1():
142     total_nitrates = 2093
143     import_ = 10
144     land = 23
145     nitrate_import = import_ * 4.5
146
147     total_nitrates += nitrate_import
148     wfsr = total_nitrates / land
149     wfsr = round(wfsr,2)
150
151     assert wfsr == 92.96
152
153 def test_import_manure_nitrates_2():
154     total_nitrates = 2093
155     import_ = 20
156     land = 50
157     nitrate_import = import_ * 4.5
158
159     total_nitrates += nitrate_import
160     wfsr = total_nitrates / land
161     wfsr = round(wfsr,2)
162
163     assert wfsr == 43.66
164
165 def test_slurry_export_nitrates_1():
166     total_nitrates = 2093
167     export = 10 * 5
168     total_nitrates -= export
169
170     wfsr = total_nitrates / 23
171     wfsr = round(wfsr,2)
172
173     assert wfsr == 88.83
174
175 def test_slurry_export_nitrates_2():
176     total_nitrates = 2093
177     export = 30 * 5
178     total_nitrates -= export
179
180     wfsr = total_nitrates / 23
181     wfsr = round(wfsr,2)

```

Figure 21: Test.py

```

142     total_nitrates = 2093
143     import_ = 10
144     land = 23
145     nitrate_import = import_ * 4.5
146
147     total_nitrates += nitrate_import
148     wfsr = total_nitrates / land
149     wfsr = round(wfsr,2)
150
151     assert wfsr == 92.96
152
153 def test_import_manure_nitrates_2():
154     total_nitrates = 2093
155     import_ = 20
156     land = 50
157     nitrate_import = import_ * 4.5
158
159     total_nitrates += nitrate_import
160     wfsr = total_nitrates / land
161     wfsr = round(wfsr,2)
162
163     assert wfsr == 43.66
164
165 def test_slurry_export_nitrates_1():
166     total_nitrates = 2093
167     export = 10 * 5
168     total_nitrates -= export
169
170     wfsr = total_nitrates / 23
171     wfsr = round(wfsr,2)
172
173     assert wfsr == 88.83
174
175 def test_slurry_export_nitrates_2():
176     total_nitrates = 2093
177     export = 30 * 5
178     total_nitrates -= export
179
180     wfsr = total_nitrates / 23
181     wfsr = round(wfsr,2)
182
183     assert wfsr == 84.48
184
185 def test_manure_export_nitrates_1():
186     total_nitrates = 2093
187     export = 30 * 5
188     total_nitrates -= export
189
190     wfsr = total_nitrates / 23
191     wfsr = round(wfsr,2)
192
193     assert wfsr == 84.48
194
195 def test_manure_export_nitrates_2():
196     total_nitrates = 2093
197     export = 30 * 5
198     total_nitrates -= export
199
200     wfsr = total_nitrates / 23
201     wfsr = round(wfsr,2)
202
203     assert wfsr == 84.48

```

Figure 22:Test.py

## Forms.py

```
11
12 class GrasslandForm(forms.Form):
13     """
14     This form will take in the farmers personal information
15     """
16     farmer_name = forms.CharField(
17         max_length=30, widget=forms.TextInput(attrs={"class": "formclass"})
18     )
19     farmer_address_line_1 = forms.CharField(
20         max_length=30,
21         required=True,
22         widget=forms.TextInput(attrs={"class": "formclass"}),
23     )
24     farmer_address_line_2 = forms.CharField(
25         max_length=30, widget=forms.TextInput(attrs={"class": "formclass"})
26     )
27     farmer_address_line_3 = forms.CharField(
28         max_length=30, widget=forms.TextInput(attrs={"class": "formclass"})
29     )
30     date = forms.DateField(
31         initial=datetime.date.today,
32         widget=forms.widgets.DateInput(attrs={"type": "date"}),
33     )
34     # date = forms.DateField(widget = forms.TextInput(attrs={ "class": "formclass"}))
35     # date.input_formats = "%d-%m-%Y"
36     county = forms.CharField(
37         label="Please select a County ",
38         widget=forms.Select(
39             choices=counties, attrs={"class": "formclass", "style": "width:276px"}
40         ),
41         max_length=1,
42     )
43     herd_no = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
44
45
46 class Grassland2(forms.Form):
47     """
48     This form will take in the farmers land information
49     """
50     owned_land__hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
51     rented_land__hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
52     time_rented__months = forms.IntegerField(
53         widget=forms.TextInput(attrs={"class": "formclass"})
54     )
55     total_tillage_area__hectares = forms.FloatField(
56         widget=forms.TextInput(attrs={"class": "formclass"})
57     )
58     area_reseeded__hectares = forms.FloatField(
59         widget=forms.TextInput(attrs={"class": "formclass"})
60     )
61
62
63 class Grassland3(forms.Form):
64     """
65     This form will take in the farmers information for a fertilizer plan, this is a new feature to be
66     implemented in the future
67     """
68
69     sample_code = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
70     date_taken = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
71     sample_area = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
72     ph = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
73     lime_required = forms.CharField(
74         widget=forms.TextInput(attrs={"class": "formclass"})
```

Figure 23: Forms.py

```

46 class Grassland2(forms.Form):
47     """
48     This form will take in the farmers land information
49     """
50     owned_land__hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
51     rented_land__hectares = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
52     time_rented__months = forms.IntegerField(
53         widget=forms.TextInput(attrs={"class": "formclass"})
54     )
55     total_tillage_area__hectares = forms.FloatField(
56         widget=forms.TextInput(attrs={"class": "formclass"})
57     )
58     area_reseeded__hectares = forms.FloatField(
59         widget=forms.TextInput(attrs={"class": "formclass"})
60     )
61
62
63 class Grassland3(forms.Form):
64     """
65     This form will take in the farmers information for a fertilizer plan, this is a new feature to be
66     implemented in the future
67
68     """
69     sample_code = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
70     date_taken = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
71     sample_area = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
72     ph = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
73     lime_required = forms.CharField(
74         widget=forms.TextInput(attrs={"class": "formclass"})
75     )
76     p_value = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
77     k_value = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
78     soil_type = forms.CharField(widget=forms.TextInput(attrs={"class": "formclass"}))
79
80
81 class Grassland4(forms.Form):
82     """
83     This form will take in the farmers feed information for livestock
84     """
85     number_compound = forms.IntegerField(
86         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
87     )
88     number_wheat = forms.IntegerField(
89         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
90     )
91     number_maize = forms.IntegerField(
92         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
93     )
94     number_maize_germ = forms.IntegerField(
95         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
96     )
97     number_oats = forms.IntegerField(
98         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
99     )
100     number_beat_pulps_molassed = forms.IntegerField(
101         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
102     )
103     number_beat_pulp_unmolassed = forms.IntegerField(
104         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
105     )
106     number_citrus_pulp = forms.IntegerField(
107         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
108     )
109     number_maize_distiller = forms.IntegerField(
110         widget=forms.TextInput(attrs={"class": "formclass"}), initial=0

```

Figure 24: Forms. py

```

149
150 class Grassland5(forms.Form):
151     """
152     This form will take in the farmers livestock information
153     """
154     def __init__(self, *args, **kwargs):
155         super().__init__(*args, **kwargs)
156
157         number_dairy_cows = forms.IntegerField(
158             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
159         )
160         number_suckler_cows = forms.IntegerField(
161             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
162         )
163         number_cattle1 = forms.IntegerField(
164             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
165         )
166         number_cattle2 = forms.IntegerField(
167             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
168         )
169         number_cattle3 = forms.IntegerField(
170             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
171         )
172         number_mountain_ewe = forms.IntegerField(
173             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
174         )
175         number_lowland_ewe = forms.IntegerField(
176             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
177         )
178         number_mountain_hogget = forms.IntegerField(
179             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
180         )
181         number_lowland_hogget = forms.IntegerField(
182             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
183         )
184         number_goats = forms.IntegerField(
185             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
186         )
187         number_horse1 = forms.IntegerField(
188             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
189         )
190         number_horse2 = forms.IntegerField(
191             widget=forms.TextInput(attrs={"class": "formclass"}), initial=0
192         )
193         farmer_name = forms.CharField(
194             widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
195         )
196
197
198 class import_Export(forms.Form):
199     """
200     This form will take in the farmers Import Export information
201     """
202     CHOICES = (("Import", "Import"), ("Export", "Export"))
203     option = forms.ChoiceField(choices=CHOICES)
204     farmer_name = forms.CharField(
205         widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
206     )
207     farmyard_manure__tonnes = forms.FloatField(
208         widget=forms.TextInput(attrs={"class": "formclass"})
209     )
210     slurry__tonnes = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
211

```

Figure 25: Forms.py

```

197
198 class import_Export(forms.Form):
199     """
200     This form will take in the farmers Import Export information
201     """
202     CHOICES = (("Import", "Import"), ("Export", "Export"))
203     option = forms.ChoiceField(choices=CHOICES)
204     farmer_name = forms.CharField(
205         widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
206     )
207     farmyard_manure__tonnes = forms.FloatField(
208         widget=forms.TextInput(attrs={"class": "formclass"})
209     )
210     slurry__tonnes = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
211
212
213 class storage(forms.Form):
214     """
215     This form will take in the farmers storage information
216     """
217     farmer_name = forms.CharField(
218         widget=forms.TextInput(attrs={"autocomplete": "off", "list": "farmers"})
219     )
220     TYPE = (("Slurry", "Slurry"), ("Farmyard Manure", "Farmyard Manure"))
221     choice = forms.ChoiceField(choices=TYPE)
222     CHOICES = (("Indoor", "Indoor"), ("Outdoor", "Outdoor"))
223     option = forms.ChoiceField(choices=CHOICES)
224     length = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
225     breadth = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
226     height = forms.FloatField(widget=forms.TextInput(attrs={"class": "formclass"}))
227     add_another_container = forms.BooleanField(required=False)
228
229     def clean(self):
230         cleaned_data = super(GrasslandForm, self).clean()
231         cleaned_data1 = super(Grassland2, self).clean()
232         cleaned_data2 = super(Grassland3, self).clean()
233         cleaned_data3 = super(Grassland4, self).clean()
234         cleaned_data4 = super(Grassland5, self).clean()
235         cleaned_data5 = super(storage, self).clean()
236         name = cleaned_data.get("farmer_name")
237         farmer_email = cleaned_data.get("farmer_email")
238         farmer_address_line_1 = cleaned_data.get("farmer_address_line_1")
239         farmer_address_line_2 = cleaned_data.get("farmer_address_line_2")
240         farmer_address_line_3 = cleaned_data.get("farmer_address_line_3")
241         date = cleaned_data.get("date")
242         herd_no = cleaned_data.get("herd_no")
243
244         if (
245             not name
246             and not farmer_email
247             and not farmer_address_line_1
248             and not farmer_address_line_2
249             and not farmer_address_line_3
250             and not date
251             and not herd_no
252         ):
253             raise forms.ValidationError("These fields are required")

```

Figure 26: Forms.py

# Templates

```
1 {% load static %}
2
3 <html>
4
5 <head>
6     {% block imag %}
7     
8     <h1> Advisors Application</h1>
9     {% endblock %}
10    <link rel="stylesheet" href="{% static 'css/styles.css' %}" />
11
12    {% block nav %}
13    <ul>
14        <li><a href="{% url 'home' %}">Home</a></li>
15        <li><a href="{% url 'storage' %}">Storage</a></li>
16        <li><a href="{% url 'importExport' %}">Import / Export</a></li>
17        <li><a href="{% url 'conductGrasslandAssessment' %}">Conduct Assessment</a></li>
18        <li><a href="{% url 'updateLsu' %}">Update LSU</a></li>
19        <li><a href="{% url 'logout' %}">Logout</a></li>
20        <li style="float: right"><a href="{% url 'records' %}">Farm Records</a></li>
21    </ul>
22    {% endblock %}
23 </head>
24
25 <body>
26
27
28     {% block body %}
29
30     {% endblock %}
31 </body>
32
33 </html>
```

Figure 27: Templates.py

```

1 {% extends "base.html" %}
2 {% include "registration/login.html" %}
3
4 {% load static %}
5
6 {% block body %}
7 <style>
8     .invalid {
9         border: 3px solid red !important;
10    }
11 </style>
12 <script>
13     window.onload = function validate_field() {
14         var elem = document.getElementById("land_table");
15         let element = rows[i].children[3].children[0];
16         if (!isInt(element.value)) {
17             element.classList.add("invalid")
18         }
19         else {
20             element.classList.remove("invalid");
21         }
22
23     function valid_fields() {
24         var valid = true;
25         var rows = document.getElementById("#id_herd_no");
26         rows[i].children[3].children[0].onkeyup = () => {
27             let element = rows[i].children[3].children[0];
28             if (!isInt(element.value)) {
29                 element.classList.add("invalid")
30             }
31             else {
32                 element.classList.remove("invalid");
33             }
34         }
35     }
36 </script>
37
38
39 <script>
40     $(function () {
41         $("#id_date").datepicker();
42     });
43 </script>
44
45 <form action="/conductGrasslandAssessment" method="POST" onsubmit="return valid_fields()" class="formindent">
46     {% csrf_token %}
47     <table border=1 id="land_table" onload="validate_field()">
48         {{ form }}
49     </table>
50
51     <div class="input-group-append" data-target="#datetimepicker1" data-toggle="datetimepicker">
52         <div class="input-group-text"><i class="fa fa-calendar"></i></div>
53     </div>
54 </div>
55 <div id="button_wrapper">
56     <input type='submit' value="Submit">
57     <input type='reset' value="Clear">
58 </div>
59 </form>
60
61 {% endblock %}

```

Figure 28: Templates.py



```

1 {% extends "base.html" %}
2 {% include "registration/login.html" %}
3
4 {% block body %}
5 <style>
6     td {
7         text-align: center;
8     }
9
10    th {
11        text-align: center;
12    }
13 </style>
14 <form action="/grasslandAssessmentResult" method="POST" class="formindent">
15     {% csrf_token %}
16     <table border="1" id="table">
17         <tr>
18             <th scope="col">Total Nitrates</th>
19             <th scope="col">Total Phosphates</th>
20             <th scope="col">Total Land Area</th>
21             <th scope="col">Grassland Stocking Rate</th>
22             <th scope="col">Wholefarm Stocking Rate</th>
23             <th scope="col">Livestock unit per Hectacre</th>
24         </tr>
25         {% for total in list_for_result %}
26         <tr>
27             <td>{{ total.0 }}</td>
28             <td>{{ total.1 }}</td>
29             <td>{{ total.2 }}</td>
30             <td>{{ total.3 }}</td>
31             <td>{{ total.4 }}</td>
32             <td>{{ total.5 }}</td>
33         </tr>
34         {% endfor %}
35     </table>
36     <div id="button_wrapper">
37         <a href="/" id="button_wrapper">
38             <input type="button" value="Home" />
39         </div>
40 </form>
41 {% endblock %}

```

Figure 29: Templates.py

```

8     border: 3px solid red !important;
9     }
10  </style>
11  <script>
12      window.onload = function validate_field() {
13          var rows = document.getElementById("land_table").rows;
14          for (let i = 1; i < rows.length; i++) {
15              var isInt = (x) => {
16                  return !isNaN(x) && !isNaN(parseInt(x));
17              };
18              rows[i].children[1].children[0].onkeyup = () => {
19                  let element = rows[i].children[1].children[0];
20                  if (!isInt(element.value)) {
21                      element.classList.add("invalid")
22                  }
23                  else {
24                      element.classList.remove("invalid");
25                  }
26              }
27          }
28      }
29
30      function valid_fields() {
31          var valid = true;
32          var rows = document.getElementById("land_table").rows;
33          for (let i = 0; i < rows.length; i++) {
34              if (rows[i].children[1].children[0].classList.contains("invalid")) {
35                  valid = false;
36                  break;
37              }
38          }
39          return valid;
40      }
41  </script>
42
43  <datalist id="farmers">
44      {% for name in farmer_list %}
45      <option value="{{ name }}">
46          {% endfor %}
47  </datalist>
48  <!-- <p>Select Farmer</p>
49      <input list="farmers" name="farmer_name" autocomplete="off"> -->
50  <br>
51  <form action="/importExport" method="POST" onsubmit="return valid_fields()" class="formindent">
52      {% csrf_token %}
53      <table border=1 id="land_table" onload="validate_field()">
54
55
56          {{ form }}
57      </table>
58      <div id="button_wrapper">
59          <input type='submit' value="Submit">
60          <input type='reset' value="Clear">
61      </div>
62  </form>
63  {% endblock %}

```

Figure 30: Templates.py